

48

LE JOURNAL QUI PEDALE FORTH

DECEMBRE 1988

МИР

HOME SWEET HOME

VOLKOV, PEDALE PLUS FORTH, SINON DANS UN AN ON Y EST ENCORE

EDITORIAL

Ce mois-ci JEDI semble avoir un sommaire amaigri. Mais question contenu, ces rubriques occupent quand même 26 pages. Et si la place occupée par la rubrique VERSION ORIGINALE est importante, elle est en plus remarquable, car pour la première fois, un article concernant TURBO-Forth est écrit et diffusé dans une revue étrangère avant de paraître dans JEDI: "Forth on the IBM: prime numbers", publié en septembre 1988 dans COMPUSER et dont l'auteur nous en a fait parvenir une copie. Et si TURBO-Forth provoque autant d'intérêt, en France et à l'étranger, nous le devons à la collaboration de tous ceux qui ont oeuvré pour le développement de sa bibliothèque de programmes et d'utilitaires ainsi que sa traduction en anglais et en allemand.

Je proposerai le mois prochain un très court programme permettant l'écriture de programmes trilingues et vous inviterai à l'utiliser largement afin que nos programmes cassent les barrières linguistiques dans lesquelles, nous Européns, nous nous enfermons.

M. PETREMANN

SOMMAIRE

Nos coordonnées: ASSOCIATION JEDI 17, rue de la Lancette, 75012 PARIS
Tel président: (1) 43.40.96.53
Tel secretaire: (1) 49.85.63.67 ou 3615 SAM*JEDI bal SECRETAIRE
TELETEL 3: 3615 SAM*JEDI (Forum, BALs, Téléchargement)

INITIATION A LA GESTION DES FICHIERS SEQUENTIELS

par Andre CHERAMY

Systéme: TURBO-Forth 83-Standard
Adaptabilité: fortement dépendant de TF83 et MSDOS
Diffusion: module M6 et téléchargement 3615 SAM*JEDI

Vous trouverez, ci-joint, un programme TURBO F83 d'initiation à la gestion des fichiers séquentiels sous TURBO F83. Ce programme propose une quarantaine de mots permettant d'ouvrir, lire, écrire, remplacer, copier, dumper, ajouter, translater, etc...

Tel qu'il se présente actuellement, ce programme est peut-être un peu long pour être publié dans JEDI. Je vous laisse le soin d'eliminer les mots qui ne vous sembleront pas nécéssaires. Ce n'est pas de la haute programmation, mais cela m'a été utile pour adapter d'anciens fichiers de références bibliographiques constitués sous RT11 (KED de DEC) à un nouveau logiciel sous MS-DOS (Framework II).

```
\  -- FORTH -----------------------------------------------------------------
\ :                 QFILES.FTH                                               :
\ :                 Initiation à la gestion des fichiers séquentiels         :
\ :                 pour  TURBO-Forth                                        :
\  -------------------------------------------------------------------------

\ QFILES.FTH par André Chéramy.   Voir explications générales après EOF.

\ Mots complémentaires pour gérer des fichiers en accés séQuentiel.

ECHO OFF

VARIABLE  HANDLE-Q        \ mémorise le ticket du fichier séquentiel

2VARIABLE  SIZE           \ longueur du fichier séquentiel

2VARIABLE  POS            \ position du pointeur du fichiers séquentiel

: BUFFER-Q   ( --- adr ) \ comme BUFFER mais pour le fichier séquentiel
  -1024 LBUF              \ ne pas oublier d'utiliser COUNT pour
  HANDLE-Q @              \ transformer cette adresse implicite en adresse
  1+ * - ;                \ explicite ( adr --- adr+1 l ) lorsque l'on
                          \ veut référer au buffer comme à une chaine

: LINE#-Q    ( --- adr ) \ comme LINE# mais pour le fichier séquentiel
  BUFFER-Q 1+ C/L + ;     \ adresse de BUFFER-Q + 1 + 255

: PATHNAME-Q ( --- adr ) \ adresse du path/filename du fichier séquentiel
  LINE#-Q 2+ ;            \ situé 2 octets plus loin dans BUFFER-Q

: ?FILE-Q          ( --- ) \ vérifie qu'un fichier séquentiel est ouvert
  HANDLE-Q @ 0=           \ ticket du fichier séquentiel non nul
  #FILES 0= OR            \ ou tous fichiers fermés par erreur système
  ABORT" pas de fichier ouvert en accès séquentiel" ;

: TOPOS      ( 2pos --- ) \ place le pointeur fichier à la position indiquée
  ?FILE-Q                 \ vérifie qu'un fichier séquentiel est ouvert
  2 ?ENOUGH               \ assure qu'au moins 2 cellules sont sur pile
  2DUP 0 0 D<             \ teste si position indiquée est < 0 ou > 2147483647
  IF 2DROP 0 0 THEN       \ si oui remplace par position 0
  2DUP SIZE 2@ D>         \ teste si > taille actuelle (limitée à 2147483647)
  IF 2DROP SIZE 2@ THEN   \ si c'est le cas remplace par taille
  HANDLE-Q @ 0 (SEEK)     \ par déplacement à partir du début
  ?DOS-ERR                \ teste si opération réussie
  POS 2! ;                \ sauve nouvelle position (sur 32 bits)

: ?POS             ( --- ) \ calcule et sauve la valeur du pointeur de position
  ?FILE-Q                 \ vérifie qu'un fichier séquentiel est ouvert
  0 0                     \ déplacement à éffectuer nul
  HANDLE-Q @ 1 (SEEK)     \ retourne la valeur du pointeur sans déplacer
  ?DOS-ERR                \ teste si opération réussie
  POS 2! ;                \ sauve nouvelle position (sur 32 bits)

: ?SIZE            ( --- ) \ calcule la longueur du fichier séquentiel
  ?POS                    \ sauve la position actuelle du pointeur
  0 0 HANDLE-Q @ 2        \ calcule la longeur du fichier
  (SEEK)  ?DOS-ERR        \ puis teste si opération réussie
  SIZE 2!                 \ sauve la longueur du fichier
  POS 2@ TOPOS ;          \ restaure la position antérieure du pointeur


: ?EOF-Q       ( --- fl ) \ indique si la fin du fichier est atteinte
  ?SIZE                   \ calcule et sauve SIZE et POS
  SIZE 2@ POS 2@ D= ;     \ si identique flag fin de fichier est vrai

: POS+             ( --- ) \ incrémente le pointeur du fichier séquentiel
  ?POS POS 2@             \ calcule et empile la position actuelle
  1 0 D+                  \ incrémente
  TOPOS ;                 \ place le pointeur fichier à la position indiquée

: POS-             ( --- ) \ décrémente le pointeur du fichier séquentiel
  ?POS POS 2@             \ calcule et empile la position actuelle
  1 0 D-                  \ décrémente
  TOPOS ;                 \ place le pointeur fichier à la position indiquée

: REWIND           ( --- ) \ positionne le pointeur au début du fichier
  0 0 TOPOS               \ position à atteindre (nombre d'octets 32 bits)
  LINE#-Q OFF ;           \ remise à zéro du No de ligne
```

```
: TOEND           ( --- ) \ positionne le pointeur en fin de fichier
  ?FILE-Q                 \ vérifie qu'un fichier séquentiel est ouvert
  0 0                     \ déplacement à effectuer (nombre d'octets 32 bits)
  HANDLE-Q @              \ ticket du fichier séquentiel
  2 (SEEK)                \ déplace pointeur à partir de la fin
  ?DOS-ERR                \ teste si opération réussie
  POS 2! ;                \ sauve la position du pointeur (sur 32 bits)

: CLOSE-Q         ( --- ) \ ferme le fichier en accès séquentiel
  HANDLE-Q @ 0= #FILES    \ vérifie qu'un fichier séquentiel est ouvert
  0= OR IF EXIT THEN      \ si ce n'est pas le cas quitte sans rien faire
  HANDLE-Q DUP            \ variable contenant le ticket du fichier
  @ (CLOSE) OFF ;         \ ferme fichier et RAZ du ticket

HEX CODE (USING) \ emprunté à Michel Zupan détermine l'attribut d'un fichier
    DX POP CX POP 4301 # AX MOV 21 INT U>= IF 0 # AX MOV THEN 1PUSH END-CODE
    DECIMAL \ 0 = ouverture lecture + écriture  1 = ouverture lecture seule)

: OPEN-Q          ( --- ) \ ouvre ou crée le fichier séquentiel dont le
                          \ path\filename suit, sans ajouter EXT$
  CLOSE-Q                 \ fermeture éventuelle si un autre déjà ouvert
  ?OPEN                   \ assure qu'un fichier suppl. peut être ouvert
  PATHWAY                 \ codifie le path\filename sans ajouter EXT$
  DUP 0 (SEARCH0)         \ cherche le fichier indiqué dans le répertoire
  IF 0 OVER (USING)       \ si existe autorise ouverture en lecture + écriture
     ?DOS-ERR 2 (OPEN)    \ si ok ouvre en lecture + écriture
  ELSE 0 (CREATE)         \ sinon en crée un nouveau en écriture
  THEN ?DOS-ERR           \ teste si opération réussie
  DUP HANDLE-Q !          \ copie et sauve ticket du fichier ouvert
  0 0 ROT 2 (SEEK)        \ calcule la longeur du fichier
  ?DOS-ERR                \ teste si opération réussie
  SIZE 2!                 \ sauve la longueur du fichier
  HERE COUNT              \ empile adr et l du path/filename indiqué
  PATHNAME-Q DUP          \ empile et duplique adr de destination
  C/PATH BLANK            \ RAZ tampon du path/filename
  SWAP CMOVE              \ transfère chaine path/filename
  REWIND ;                \ repositionne pointeur et LINE#-Q

: .LINE-Q         ( --- ) \ affiche ligne courante présente dans BUFFER-Q
  CR BUFFER-Q COUNT       \ conversion chaine implicite en chaine explicite
  -TRAILING TYPE ;        \ et affichage


: .LINE#-Q        ( --- ) \ affiche le No de ligne courante présent dans le
  LINE#-Q @ . ;           \ buffer du fichier séquentiel

: .PATHNAME-Q     ( --- ) \ affiche le path/filename du fichier séquentiel
  PATHNAME-Q C/PATH       \ adr et lmax du tampon de path/filename
  -TRAILING TYPE ;        \ affichage de la chaine proprement dite

: .POS            ( --- ) \ affiche la valeur du pointeur de position
  POS 2@ UD. ;            \ du fichier séquentiel

: .SIZE           ( --- ) \ affiche la longueur du fichier séquentiel
  SIZE 2@ UD. ;           \ valeur maxi 2147483647 (voir TOPOS)

: PUT-$      ( adr l -- ) \ la chaine explicite indiquée sera écrite dans le
                          \ fichier à la position indiquée par le pointeur
  ?FILE-Q                 \ teste si fichier séquentiel ouvert
  2 ?ENOUGH               \ assure qu'au moins 2 cellules sont sur pile
  DSEGMENT -ROT           \ place la valeur du segment en 3ème position
  HANDLE-Q @              \ empile le ticket du fichier
  (PUT) ?DOS-ERR          \ écrit et teste si opération réussie
  DROP ?SIZE              \ mise à jour de SIZE, longueur du fichier
  ?POS ;                  \ mise à jour de POS,  position du pointeur

: PUT-C         ( o --- ) \ le caractère indiqué sera écrit dans le fichier
                          \ séquentiel à la position indiquée par le pointeur
  ?FILE-Q                 \ teste si fichier séquentiel ouvert
  1 ?ENOUGH               \ assure qu'au moins 1 cellule est présente sur pile
  BUFFER-Q     1 OVER !   \ place 1 en début de buffer (longueur)
  1+ !                    \ incrémente adresse et  y place l'octet
  DSEGMENT                \ empile la valeur du segment puis
  BUFFER-Q COUNT          \ adresse et longueur de chaine (1 caractère)
  HANDLE-Q @              \ empile le ticket du fichier séquentiel
  (PUT) ?DOS-ERR          \ écrit et teste si opération réussie
  DROP ?SIZE              \ mise à jour de SIZE, longueur du fichier
  ?POS ;                  \ mise à jour de POS,  position du pointeur

: PUT-L      ( adr l -- ) \ la chaine explicite indiquée sera écrite dans
  PUT-$                   \ le fichier séquentiel ainsi que CR et LF à la
  13 PUT-C 10 PUT-C ;     \ position indiquée par le pointeur

: APPEND-$   ( adr l -- ) \ teste si fichier séquentiel est ouvert, positionne
  TOEND PUT-$ ;           \ pointeur à la fin et ajoute la chaine indiquée

: APPEND-C      ( o --- ) \ teste si fichier séquentiel est ouvert, positionne
  TOEND PUT-C ;           \ pointeur à la fin et ajoute le caractère indiqué

: APPEND-L   ( adr l -- ) \ ajoute la chaine explicite indiquée ainsi que
  TOEND PUT-L ;           \ CR et LF à la fin du fichier séquentiel
```

```
: GETLINE-Q         ( --- ) \ saisit une ligne dans le fichier séquentiel
  ?EOF-Q IF EXIT THEN       \ teste si fin de fichier déjà atteinte
  BUFFER-Q                  \ empile adr du fichier séquentiel puis transfère
  HANDLE-Q @ (GETLINE)      \ tous les caractères jusqu'à CR vers BUFFER-Q
  DROP                      \ élimine le flag généré par (GETLINE) pour EOF?
  LINE#-Q 1+! ECHO @        \ incrémente le numéro de ligne courante, teste ECHO
  IF .LINE-Q THEN           \ si ECHO ON affiche la ligne
  ?POS ;                    \ met à jour le pointeur utilisateur

: SKIP-Q          ( n --- ) \ saute n lignes dans le fichier ( n>0 )
  ?FILE-Q                   \ teste si fichier séquentiel ouvert
  1 ?ENOUGH                 \ assure qu'au moins 1 cellule est présente sur pile
  DUP 0=                    \ teste si n est nul
  IF DROP EXIT THEN         \ si oui sort sans rien sauter
  ECHO @ >R ECHO OFF        \ empile l'état de ECHO avant invalidation
  0 DO                      \ tente d'effectuer n sauts de ligne
     ?EOF-Q                 \ teste d'abord si fin de fichier
     IF LEAVE               \ si oui quitte boucle sans rien sauter
     ELSE GETLINE-Q THEN    \ si non transfert une ligne dans buffer
    LOOP                    \ sort si n transferts ou si fin de fichier
  R> ECHO ! ;               \ restaure la valeur initiale de echo

: TOLINE          ( n --- ) \ positionne au début de ligne No n ( n>0 )
  1 ?ENOUGH                 \ assure qu'au moins 1 cellule est présente sur pile
  DUP 0=                    \ teste si n est nul
  IF DROP EXIT THEN         \ si oui sort sans rien faire
  REWIND                    \ remise à zéro du pointeur et de LINE#-Q
  1- SKIP-Q ;               \ début = ligne 0 pour LINE#-Q mais 1 pour TOLINE

: <--L              ( --- ) \ positionne le pointeur au début ligne précédente
  LINE#-Q @ TOLINE ;        \ n'est valable que si LINE#-Q est à jour

: L-->              ( --- ) \ positionne le pointeur au début de ligne suivante
  1 SKIP-Q ;                \ simple non?

: REPLACE-L ( adr l n -- )  \ remplacement sécurisé de la ligne n (n positif)
  ?FILE-Q                   \ teste si fichier séquentiel ouvert
  3 ?ENOUGH                 \ assure qu'au moins 3 cellules sont sur la pile
  OVER 0=                   \ teste longueur de la chaine
  IF 2DROP DROP EXIT THEN   \ chaine vide interdite
  REWIND                    \ remise à zéro du No de ligne et du pointeur
  DUP SKIP-Q                \ duplique No et écrit ligne cible dans buffer
  LINE#-Q @ <>              \ compare No ligne cible et No ligne trouvée
  IF APPEND-L EXIT THEN     \ si fichier trop court, ajoute ligne à la fin
  BUFFER-Q C@ 1-            \ cherche longueur de la ligne cible
  2DUP SWAP - >R            \ sauve différence longueur cible longueur source
  MIN                       \ longueur transférable sans écrasement
  <--L PUT-$                \ écrit chaine au début de ligne précédente
  R> DUP 0>                 \ récupère et teste diff long cible long source
  IF 0 DO 32 PUT-C LOOP     \ si cible > source complète avec espaces
  THEN ;

: GET-L       ( adr l --)   \ affecte ligne du fichier séquentiel à une chaine
  ?FILE-Q                   \ teste si fichier séquentiel ouvert
  2 ?ENOUGH                 \ assure qu'au moins 2 cellules sont sur  pile
  ?EOF-Q IF 2DROP EXIT THEN    \ teste si fin de fichier déjà atteinte
  2DUP BLANK                \ remplit d'espaces la chaine explicite indiquée
  1 SKIP-Q                  \ transfert d'une ligne vers BUFFER-Q sans echo
  BUFFER-Q COUNT            \ adresse explicite de cette ligne dans BUFFER-Q
  2SWAP                     \ (adr2 l2 adr1 l1 --- adr1 l1 adr2 l2)
  ROT                       \ (adr1 l1 adr2 l2 --- adr1 adr2 l2 l1)
  MIN                       \ (adr1 adr2 l2 l1 --- adr1 adr2 lmin)
  MOVE ;                    \ tranfert (partiel si place insuffisante)

: GET-$     ( adr l --- )   \ lit l caractères dans le fichier à partir du
                            \ pointeur et les copie dans la chaine indiquée
  2 ?ENOUGH                 \ assure qu'au moins 2 cellules sont sur pile
  ?EOF-Q IF 2DROP EXIT      \ teste si fin de fichier déjà atteinte
  THEN 2DUP BLANK           \ remplit d'espaces la chaine explicite indiquée
  DSEGMENT -ROT             \ place la valeur du segment en 3ème position

  HANDLE-Q @                \ ticket du fichier séquentiel
  (GET) ?DOS-ERR DROP       \ lit la chaine et teste si opération réussie
  ?POS ;                    \ mise à jour de POS,  position du pointeur

: GET-C           ( --- c ) \ lit le caractère situé au pointeur du fichier
                            \ séquentiel, teste si fichier séquentiel ouvert
  ?EOF-Q IF 26 EXIT THEN    \ si fin de fichier retourne ^Z
  DSEGMENT BUFFER-Q 1+      \ empile valeur du segment, adresse reelle buffer,
  1 HANDLE-Q @              \ longueur de chaine (1 car) et ticket du fichier
  (GET) ?DOS-ERR DROP       \ lit le caractère et teste si opération réussie
  BUFFER-Q 1+ C@            \ adresse réelle de chaine implicite
  ?POS ;                    \ mise à jour de POS,  position du pointeur

: EOF-Q             ( --- ) \ force fin du fichier à la position pointeur
\ 26 PUT-C                  \ facultatif, écrit un ^Z à position courante
  BUFFER-Q 0                \ chaine quelconque mais de longueur nulle
  PUT-$ ;                   \ écrit chaine vide à la position courante

: TOSIZE     ( 2len --- )   \ augmente taille fichier jusqu'à la valeur indiquée
  ?FILE-Q                   \ vérifie qu'un fichier séquentiel est ouvert
  2 ?ENOUGH                 \ assure qu'au moins 2 cellules sont sur pile
  ?POS POS 2@ >R >R         \ sauve la position actuelle du pointeur
  2DUP 0 0 D<               \ teste si valeur indiquée est < 0 ou > 2147483647
  IF 2DROP EXIT THEN        \ si oui garde la taille actuelle
```

```
    SIZE 2@ DMAX              \ nouvelle taille > ou = taille précédente
    HANDLE-Q @ 0 (SEEK)       \ envoie le pointeur à la nouvelle fin de fichier
    ?DOS-ERR 2DROP            \ teste si opération réussie
    EOF-Q                     \ seul (PUT) valide la taille, écrit chaine vide
    R> R> TOPOS ;             \ restaure la position antérieure du pointeur

  HEX
  : DUMP-Q          ( --- ) \ affiche le contenu hexa et ascii du fichier dont
    OPEN-Q CR HEX            \ le "path\filename" suit
    BEGIN                    \ après ouverture, commence une boucle
     ?EOF-Q IF CLOSE-Q       \ est-ce la fin du fichier? si oui ferme,
     DECIMAL EXIT THEN       \ restaure mode décimal et quitte
     DSEGMENT BUFFER-Q 1+    \ valeurs du segment, du début réel du buffer, du
     10 HANDLE-Q @ (GET)     \ nombre de caractères à lire, du ticket et on lit
     ?DOS-ERR BUFFER-Q 1+    \ ( len --- len adr ) chaine lue
     SWAP BOUNDS 2DUP        \ adresses de début et de fin de chaine
     4 SPACES                \ marge gauche du tableau affiché
     ?DO I C@ 0 <# # # #>    \ scrute un à un les caractères lus et les
         TYPE SPACE          \ convertit en chaine hexa sur deux digits
     LOOP                    \ caractère suivant jusqu'a adresse de fin
     38 #OUT @ - SPACES      \ aligne affichage pour les ascii sur 56ème colonne
     ?DO I C@ DUP 20 <       \ boucle analogue, scrute caractères un à un
         IF DROP 20 THEN     \ si CTRL remplace par un espace (car les CTRL
         EMIT                \ perturbent l'affichage) puis affiche
     LOOP CR STOP?           \ l'adresse de fin a été atteinte, on arrête?
   UNTIL CLOSE-Q DECIMAL ;   \ fermeture fichier et retour mode décimal
  DECIMAL

  : SUM-Q          ( --- S ) \ retourne la check-Qum du fichier dont le
    OPEN-Q 0                 \ "path\filename" suit, après avoir ouvert le
    BEGIN                    \ fichier, initialise une somme nulle sur pile
     ?EOF-Q                  \ teste si fin de fichier atteinte
     IF CLOSE-Q EXIT THEN    \ si oui ferme et quitte avec somme sur la pile
     DSEGMENT BUFFER-Q 1+    \ segment et adresse où le caractère lu sera mis
     1 HANDLE-Q @ (GET)      \ 1 caractère à lire, ticket du fichier, lecture
     ?DOS-ERR DROP           \ opération réussie? élimine N = 1 caractère lu

     BUFFER-Q 1+ C@ +        \ empile caractère lu, ajoute à somme précédente
     STOP?                   \ c'est trop long? on arrête?
    UNTIL CLOSE-Q ;          \ si oui ferme le fichier

  \ Les mots suivants, précédés de la lettre I concernent un fichier séquentiel
  \ en lecture seule (I comme Input ou Inspect). Chacun d'entre eux est analo-
  \ gue au mot correspondant défini précédemment pour le fichier séquentiel en
  \ lecture + écriture (sauf mention spéciale). Ces mots permettent notamment
  \ des transferts entre fichiers (copie, conversion etc...).

  VARIABLE  IHANDLE-Q

  2VARIABLE ISIZE

  2VARIABLE IPOS

  : IBUFFER-Q -1024 LBUF IHANDLE-Q @ 1+ * - ;

  : I?FILE-Q IHANDLE-Q @ 0= #FILES 0= OR
    ABORT" pas de fichier séquentiel ouvert en lecture seule" ;

  : ITOPOS I?FILE-Q 2 ?ENOUGH 2DUP 0 0 D< IF 2DROP 0 0 THEN 2DUP ISIZE 2@ D>
    IF 2DROP ISIZE 2@ THEN IHANDLE-Q @ 0 (SEEK) ?DOS-ERR IPOS 2! ;

  : I?POS I?FILE-Q 0 0 IHANDLE-Q @ 1 (SEEK) ?DOS-ERR IPOS 2! ;

  : I?SIZE I?POS 0 0 IHANDLE-Q @ 2 (SEEK) ?DOS-ERR ISIZE 2! IPOS 2@ ITOPOS ;

  : I?EOF-Q I?SIZE ISIZE 2@ IPOS 2@ D= ;

  : IREWIND 0 0 ITOPOS ; \ attention simplifié

  : ICLOSE-Q IHANDLE-Q @ 0= #FILES 0= OR
    IF EXIT THEN IHANDLE-Q DUP @ (CLOSE) OFF ;

  : IOPEN-Q ICLOSE-Q ?OPEN PATHWAY DUP 0 (SEARCH0) IF 1 OVER (USING) ?DOS-ERR
    0 (OPEN) ELSE DROP EXIT THEN ?DOS-ERR DUP IHANDLE-Q ! 0 0 ROT 2 (SEEK)
    ?DOS-ERR ISIZE 2! IREWIND ; \ attention simplifié voir OPEN-Q

  : I.LINE-Q CR IBUFFER-Q COUNT -TRAILING TYPE ;

  : IGETLINE-Q I?EOF-Q IF EXIT THEN IBUFFER-Q IHANDLE-Q @ (GETLINE)
    DROP ECHO @ IF I.LINE-Q THEN I?POS  ; \ attention simplifié cf GETLINE-Q

  : ISKIP-Q I?FILE-Q 1 ?ENOUGH DUP 0= IF DROP EXIT THEN ECHO @ >R ECHO OFF
    0 DO I?EOF-Q IF LEAVE ELSE IGETLINE-Q THEN LOOP R> ECHO ! ;

  : ITOLINE 1 ?ENOUGH DUP 0= IF DROP EXIT THEN IREWIND 1- ISKIP-Q ;

  : IGET-L I?FILE-Q 2 ?ENOUGH I?EOF-Q IF 2DROP EXIT THEN 2DUP BLANK 1 ISKIP-Q
    IBUFFER-Q COUNT 2SWAP ROT MIN MOVE ;

  : IGET-$ 2 ?ENOUGH I?EOF-Q IF 2DROP EXIT THEN 2DUP BLANK DSEGMENT -ROT
    IHANDLE-Q @ (GET) ?DOS-ERR DROP I?POS ;

  : IGET-C I?EOF-Q IF 26 EXIT THEN DSEGMENT IBUFFER-Q 1+ 1 IHANDLE-Q @
    (GET) ?DOS-ERR DROP IBUFFER-Q 1+ C@ I?POS  ;

  \ Les mots suivants sont des exemples d'applications
  \ (conversion, translation... tout est possible!)
```

```
: FRUS          ( --- ) \ conversion ascii ISO25 (FRançais) en ascii IBM-US
                        \ syntaxe: FRUS filename-source filename-cible
  IOPEN-Q               \ ouvre fichier source dont le "path\filename" suit
  OPEN-Q                \ crée fichier cible dont le "path\filename" suit
  BEGIN IGET-C          \ boucle de saisie des caractères un à un
   CASE                 \ examine le caractère lu
   94 OF 94 PUT-C IGET-C \ si ^ on l'écrit et on lit caractère suivant (N°2)
      CASE              \ examine le N°2
        8 OF 8 PUT-C IGET-C \ si BS on l'écrit et lit caractère suivant (N°3)
          CASE           \ examine le N°3
          97  OF POS- POS- 131 ENDOF \ si a on remplace ^ et BS par â
          101 OF POS- POS- 136 ENDOF \ si e                         ê
          105 OF POS- POS- 140 ENDOF \ si i                         î
          111 OF POS- POS- 147 ENDOF \ si o                         ô
          117 OF POS- POS- 150 ENDOF \ si u                         û
          DUP ENDCASE        \ si rien trouvé pour le N°3 on le laisse tel quel
      DUP ENDCASE ENDOF      \ si le N°2 pas BS on le laisse tel quel
  126 OF 126 PUT-C IGET-C \ si tilde on écrit et lit caractère suivant (N°2)
      CASE                 \ examine le N°2
      8 OF 8 PUT-C IGET-C \ si BS on écrit et on lit caractère suivant (N°3)
        CASE               \ examine le N°3
          97  OF POS- POS- 132 ENDOF \ si a on remplace tilde et BS par ä
          101 OF POS- POS- 137 ENDOF \ si e                             ë
          105 OF POS- POS- 139 ENDOF \ si i                             ï
          111 OF POS- POS- 148 ENDOF \ si o                             ö
          117 OF POS- POS- 129 ENDOF \ si u                             ü
          DUP ENDCASE        \ si rien trouvé pour le N°3 on le laisse tel quel
      DUP ENDCASE ENDOF    \ si le N°2 pas BS on le laisse tel quel
   64 OF 133 ENDOF         \ si @ il deviendra à
   91 OF 248 ENDOF         \ si [              °
   92 OF 135 ENDOF         \ si \              ç
   93 OF  21 ENDOF         \ si ]              paragraphe
  123 OF 130 ENDOF         \ si {              é
  124 OF 151 ENDOF         \ si |              ù
  125 OF 138 ENDOF         \ si }              è
  DUP ENDCASE              \ rien n'a été trouvé on laisse tel quel
  PUT-C STOP? I?EOF-Q OR \ écrit caractère et teste si doit reboucler
  UNTIL CLOSE-Q ICLOSE-Q ;

: FRUST         ( --- ) \ idem Turbo (réduit aux + courants: à ç ê ô é è ù)
  IOPEN-Q               \ ouvre fichier source dont le "path\filename" suit
  OPEN-Q                \ crée fichier cible dont le "path\filename" suit
  BEGIN IGET-C          \ boucle de saisie des caractères un à un
   CASE                 \ examine le caractère lu
   94 OF 94 PUT-C IGET-C \ si ^ on l'écrit et on lit caractère suivant (N°2)
      CASE              \ examine le N°2
      8 OF 8 PUT-C IGET-C \ si BS on l'écrit et lit caractère suivant (N°3)
        CASE             \ examine le N°3
        101 OF POS- POS- 136 ENDOF \ si e il deviendra ê
        111 OF POS- POS- 147 ENDOF \ si o              ô
        DUP ENDCASE        \ si rien trouvé pour le N°3 on le laisse tel quel
      DUP ENDCASE ENDOF    \ si le N°2 pas BS on le laisse tel quel
   64 OF 133 ENDOF         \ si @ il deviendra à
   92 OF 135 ENDOF         \ si \              ç
  123 OF 130 ENDOF         \ si {              é
  124 OF 151 ENDOF         \ si |              ù
  125 OF 138 ENDOF         \ si }              è
  DUP ENDCASE              \ rien n'a été trouvé on laisse tel quel
  PUT-C STOP? I?EOF-Q OR \ écrit caractère et teste si doit reboucler

  UNTIL CLOSE-Q ICLOSE-Q ;

: LM+          ( n --- ) \ crée nouv fichier avec +n espaces de marge gauche
                         \ syntaxe: LM+ filename-source filename-cible
  IOPEN-Q                \ ouvre fichier source dont le "path\filename" suit
  OPEN-Q                 \ crée fichier cible dont le "path\filename" suit
  BEGIN IGETLINE-Q       \ boucle de saisie des lignes une à une
  DUP 0 DO 32 PUT-C LOOP \ écriture de n espace dans le fichier cible puis
  IBUFFER-Q DUP 1+       \ adr réelle chaine (IBUFFER-Q = chaine implicite)
  OVER @ PUT-L           \ longueur de chaine et écriture ligne dans cible
  STOP? I?EOF-Q OR       \ teste si doit reboucler
  UNTIL CLOSE-Q ICLOSE-Q ; \ simple exemple de ce qui est possible

: REPLACE-C ( o o' --- ) \ crée nouveau fichier en remplaçant caract o par o'
                         \ syntaxe: REPLACE-C filename-source filename-cible
  IOPEN-Q                \ ouvre fichier source dont le "path\filename" suit
  OPEN-Q                 \ crée fichier cible dont le "path\filename" suit
  BEGIN IGET-C           \ boucle de saisie des caractères un à un
  DUP 3 PICK =           \ compare avec le caractère à chercher
  IF  DROP DUP THEN      \ si = remplace par caractère de remplacement
  PUT-C STOP? I?EOF-Q OR \ écrit caractère et teste si doit reboucler
  UNTIL CLOSE-Q ICLOSE-Q ; \ mot très utile pour effectuer des translations

ECHO ON

EOF \ Commentaires généraux
```

Turbo F83 est tout à fait apte à la gestion des fichiers séquentiels. Mais les mots dédiés à cet usage sont éparpillés dans le vocabulaire (il en existe une quarantaine) et ils ne sont pas toujours faciles à utiliser. C'est pouquoi ce texte d'initiation a été écrit, les mots principaux regroupés et expliqués, pourquoi aussi quelques mots complémentaires sont proposés.

Une trentaine de mots très astucieux ont été définis par Michel Zupan dans UFILES.FTH, ce sont entre autres et pour exemple:

- USING? qui indique si un fichier est autorisé en lecture ou écriture
- USING qui modifie le type d'accès autorisé (vous en aurez besoin!)
- FILESIZE qui détermine la longueur d'un fichier
- OPEN-R qui ouvre ou crée un fichier en accès direct... ainsi qu'une famille de mots adaptés à la gestion de ce type de fichier
- OPEN-BLK qui ouvre un fichier-blocks du F83 Laxen et Perry... ainsi qu'une famille de mots adaptés à la gestion de ce type de fichier
- UNBLOCK pour la conversion de fichiers-blocks en fichiers TURBO-Forth.

Les mots proposés ici sont directement inspirés de ce travail de Michel Zupan paru dans JEDI. Voici tout d'abord, quelques explications.

Les fichiers DOS sont caractérisés par leur "path\filename", leur taille en octets, leurs date et heure de création et enfin leur attribut. Cet attribut est un octet dont les bits 0 à 5 sont à 1 si le fichier est autorisé en lecture seule, caché, système, volume, répertoire, modifié respectivement.

Afin de pouvoir contenir n'importe lequel des 256 caractères de l'ascii étendu, la fin des fichiers n'est pas marquée par un caractère particulier mais est reconnue lorsque la position du pointeur égale la taille du fichier.

Dans certains cas, un caractère (^Z pour exemple) ou un mot (EOF dans le cas des fichiers de Turbo F83) sont utilisés pour arrêter la saisie, la transmission ou l'interprétation.

Les fichiers sont "bufferisés" dans une zone de la mémoire haute, à raison de 322 octets par fichier, (cette valeur est calculée par LBUF). Chaque fichier est identifié par son ticket. Pour exemple, HANDLE contient le ticket du flot d'entrée.

L'adresse du buffer de chaque fichier est calculée par BUFFER selon le ticket du fichier. Pour exemple, si un fichier de ticket 5 est ouvert, son buffer sera situé de l'adresse 52580 à l'adresse 62901. Dans chaque cas, le premier octet, dont l'adresse est BUFFER, indique la longueur de la ligne, avec un maximum de 255, (cette valeur maxi est mémorisée dans C/L).

Les 255 octets suivant, dont l'adresse est BUFFER+1 servent de tampon de ligne. Le numéro de ligne courante est indiqué dans les deux octets suivants, dont l'adresse est LINE#, c'est à dire BUFFER+256. Ce numéro de ligne est informatif, il ne sert pas a positionner (en principe).

Enfin les 64 derniers octets (valeur maxi mémorisée dans C/PATH), dont l'adresse est PATHNAME, contiennent la chaine "path\filename" du fichier.

De nombreux mots de Turbo F83 sont utilisés pour manipuler les fichiers séquentiels, soit pour mimer des commandes DOS, soit pour gérer le flot d'entrée. Ce sont pour exemple: DIR, REN, LIST, OPEN, CLOSE, ALLCLOSE, INCLUDE, HANDLE, BUFFER etc...). Les primitives correspondantes sont particulièrement intéressantes. Ce sont par exemple:

FILENAME (--- adr0) qui interprète le "path\filename" qui suit et retourne l'adresse d'une chaine ascii0 utilisable par (OPEN) ou (CREATE).

PATHWAY (--- ade0) idem sans ajouter d'extension par défaut.

(SEARCH0) (adr0 att --- fl) qui cherche le fichier spécifié par adr0 et étant autorisé en lecture seule si att=1 ou en lecture écriture si att=0 et retourne un flag vrai si existe bien. Attention l'attribut de (SEARCH0), de (CREATE) et de USING est différent de l'accès de (OPEN), voir plus loin ces mots.

(OPEN) (adr0 accès --- ticket err) qui ouvre un fichier spécifié par adr0 en lecture seule si accès = 0 ou en écriture seule si accès=1 ou en lecture et écriture si accès=2 et retourne le ticket et un flag d'erreur.

?DOS-ERR (err ---) signale une éventuelle erreur. Si err=0 pas d'erreur, sinon indique la nature de l'erreur, pour exemple "accès fichier interdit" lorsque err=5.

(CREATE) (adr0 att --- ticket err) qui crée un fichier spécifié par adr0 en lecture seule si att=1 ou en lecture et écriture si att=0 et retourne le ticket et un flag d'erreur. Attention l'attribut de (CREATE), de (SEARCH0) et de USING (voir plus loin) est différent de l'accès de (OPEN).

(CLOSE) (ticket ---) qui ferme le fichier spécifié par le ticket indiqué.

(SEEK) (2len ticket way --- 2pos err) mot très important qui déplace le pointeur du fichier, indiqué par le ticket, de 2len caractères à partir de la position courante si way=1 ou à partir du début si way=0 ou à partir de la fin si way=2. En outre, et c'est le coté génial de ce mot, il retourne la nouvelle valeur du pointeur et un flag d'erreur. Mais attention ce mot n'est pas sécurisé et peut placer le pointeur n'importe où!

(PUT) (segment adr l ticket --- n err) qui effectue le transfert de la chaine explicite désignée vers le fichier pointé par le ticket et l'écrit à la position courante. Le flag indique si l'écriture a réussi et n indique le nombre de caractères réellement écrit dans le fichier.

(GET) (segment adr n ticket --- n' err) qui effectue le transfert de n caractères lus dans le fichier indiqué par le ticket et ce à partir de la position courante et recopiés à l'adresse indiquée située dans segment. Le flag indique si la lecture a réussi, n' étant le nombre de caractères réellement lus.

DSEGMENT (--- segment) retourne la valeur du segment courant du système (data-segment). Utilisable avec (PUT) et (GET).

(GETLINE) (buffer ticket --- flag) qui effectue le transfert d'une ligne complète, c'est à dire terminée par un <LF>, prise dans le fichier indiqué par le ticket, à la position courante vers une chaine implicite (copie), en général, il s'agit de celle pointée par BUFFER. Le nombre de caractères lus (maxi 255) est placé dans le premier octet de cette chaine implicite. Les caractères de contrôles sont éliminés, y compris CR et LF. Le flag indique si la fin du fichier a été atteinte (NB le fin est forcée en cas d'erreur).

Les mots complémentaires proposés ici servent essentiellement à séparer la gestion du flot d'entrée de celle du fichier séquentiel, comme l'avait fait Michel Zupan pour les fichiers à accès direct. Ces mots complémentaires sont commentés dans le détail et sont donc faciles à comprendre et à modifier. Voici quelques explications complémentaires.

Il est un peu difficile d'écrire dans un fichier en accès séquentiel, c'est pourquoi il n'y a pas, en standard, de mot du genre PUTLINE pour écrire une ligne. En effet si la nouvelle ligne n'a pas la meme longueur que celle qu'elle "écrase", la situation devient vite inextricable. L'utilisation de PUT-$, PUT-C et PUT-L est donc limitée. Une tentative a été faite avec REPLACE-L qui permet de remplacer une ligne entière du fichier par une chaine qui sera tronquée si sa longueur est supérieure à celle de la ligne d'origine ou qui sera complétée avec des espaces si elle est plus courte. L'utilité de ce mot reste à prouver. Sinon il faut s'orienter vers un programme de traitement de texte. Par contre, il est très facile d'ajouter du texte à la fin. C'est le rôle des mots complémentaires APPEND-$, APPEND-C et APPEND-L.

Lorsque le mot GET-C atteint la fin du fichier, ne pouvant rien lire et devant retourner un caractère, on a choisi de lui faire retourner ^Z (ascii 26), valeur qui est modifiable.

LINE#-Q, numéro de ligne courante est informatif. Le début du fichier correspond à la valeur zéro. Sa mise à jour est

douteuse, elle n'est pas prise en charge par TOPOS, (PUT) (GET) et (SEEK), ce qui restraint son usage. Toutefois OPEN-Q et REWIND le remettent à zéro, GETLINE-Q et ses dérivés SKIP-Q, TOLINE, GET-L et L--> l'incrémentent. <--L est basé sur LINE#-Q et n'est pas plus fiable que ce dernier. Si TOLINE est conçu pour positionner à la ligne voulue, il utilise une valeur différente de celle de LINE#-Q, en effet, le début du fichier correspond à la valeur 1 (1ère ligne). TOLINE et SKIP-Q utilisent un No de ligne positif. Toute valeur signée sera considérée comme un nombre positif. Si le No de ligne est plus grand que le nombre de lignes du fichier, le pointeur ira à la fin du fichier.

Le flag EOF? n'est pas utilisé. Toutefois un mot nouveau a été crée pour tester si la fin du fichier est atteinte, c'est ?EOF-Q.

Enfin, le travail simultané sur deux fichiers séquentiels, dont l'un est ouvert en lecture seule (protection) offre la possibilité d'effectuer des copies, conversions, adaptations diverses, translations ou élimination de caractères sans autre limitation que votre imagination.

Télématique

CONTENU DU FORUM 3615 SAM*JEDI

Préambule:
J'ai pris la décision de ne plus faire la "LETTRE DU SECRETAIRE" qui était jointe à chaque numéro de JEDI, ceci pour deux raisons:

1) permettre de gagner deux pages au profit de JEDI, question poids de la revue, qui ne doit pas dépasser, enveloppe comprise 100 grammes, sinon les frais d'expédition doublent.

2) depuis la mise en place de SAM*JEDI, notre serveur, de nombreuses informations ne nécessitent plus d'être diffusées sous forme de complément à la revue. Grâce à SAM*JEDI, les informations URGENTES peuvent être communiquées très rapidement, ce qui a été le cas notamment de l'invitation HARRIS au séminaire du 18 novembre et dont nombre d'adhérents n'auraient pu prendre connaissance par lettre du fait de la grève des postiers.

Quand à l'accès par 3615, il ne sera pas remis en cause. SAM envisage même d'ouvrir un accès à SAM*JEDI par le 3616, afin que les adhérents puissent accéder à notre serveur depuis leur lieu de travail si leur ligne est interdite de 3615. Cet accès par 3616 est en cours de négociation avec FRANCE TELECOMM.

F32 04.10.88 17h39
F32 A PRIS CONTACT AVEC HARRIS AFIN DE REGLER NOS POSITIONS RESPECTIVES SUR NOTRE DEVELOPPEMENT. HARRIS AYANT DEJA ACCES A SAM*JEDI,LES MESSAGES EMANANT DE F32 SONT DESORMAIS EMIS DIRECTEMENT VERS LES BAL DES PARTICIPANTS. DE MEME ADRESSEZ-VOUS DIRECTEMENT A F32. CECI VAUT TANT QUE NOUS N'AVONS PAS DEFINI NOS RELATIONS, CE QUI DEVRAIT PRENDRE->DECEMBRE. PLUS QUE JAMAIS CONTACTEZ F32 NOYAU 5G8D F32 HAUT LES COEURS!

FORTH7 06.10.88 14h24
SUPER EDITEUR DE LIGNE ULTRA-SIMPLE: les concepteurs de Turbo-Forth sont encore plus geniaux qu'ils le pensent: saviez-vous que les variables STRING ont exactement la structure d'un tampon MS-DOS ? Definissez le mot simplissime suivant:

```
: EXPECT$ DROP 2- 10 BDOS DROP CR ;
```

Puis essayez: 40 STRING TEST

```
TEST EXPECT$
```

Pour peu que vous ayez charge DOSEDIT ou CED etc... vous entrerez TEST en beneficiant des fleches, de l'insertion par INS, du BEEP en fin de tampon ou du rappel des precedentes entrees... ABSOLUMENT INCROYABLE NON ?

F32 06.10.88 18h22
F32 CHERCHE FORTH POUR ATARI 520ST VEUILLEZ UTILISER MA BAL SVP. MERCI D'AVANCE

TICK 10.10.88 09h24
COMMENT CREER LE MOT C: OU D:. EN F83 IL EXISTE BIEN A: ET B: MAIS LE MOT SELECT UTILSE PAR CES MOTS N'EST PAS RECONNU

```
: A:   0 SELECT ;  OK
: C:   3 SELECT ;  SELECT?
```

Y A T'IL UNE AUTRE SOLUTION QUE UN ASSIGN D=B

SECRETAIRE 10.10.88 15h44
ATTENTION, LE MOT SELECT EN F83 EST DEFINI DANS LE VOCABULAIRE DOS:

```
  ONLY FORTH DOS ALSO FORTH ALSO
  : C 2 SELECT ;   : D 3 SELECT ;
  ONLY FORTH ALSO
OU PLUS SIMPLEMENT
  : C: 2 [ DOS ] SELECT [ FORTH ] ;
```

SINON SUR TURBO-FORTH, C: D: ET E: SONT DEJA DEFINIS.

TICK 11.10.88 09h52
Votre super syst expert FORTHlog a-t-il ete adapte au turbo-FORTH notamment; si OUI son prix est il change? Si NON peut on acquerir directement la version IIB? MERCI...

SECRETAIRE 11.10.88 12h44
ADAPTATION FORTHLOG II: CE GENERATEUR DE SYSTEMES EXPERTS N'A PAS ETE ADAPTE A TURBO-FORTH (DOMMAGE). MAIS COMME VOUS POUVEZ L'IMAGINER, NOUS NE SOMMES QUE TRES PEU DE MEMBRES ACTIFS. S'IL Y A DES COURAGEUX QUI SOUHAITENT SE LANCER DANS L'AVENTURE... CEPENDANT, POUR INFO, JEDI VA DIFFUSER EN 4 PARTIES, UNE VERSION DU LANGAGE PROLOG ECRITE EN FORTH. JE PENSE QU'IL VAUT MIEUX ATTENDRE CE MERVEILLEUX ARTICLE PUIS DECIDER ENSUITE DE LA VOIE A SUIVRE. SALUTATIONS.

TICK 12.10.88 17h23
SLT LES FORTHIENS: COMME JE NE SUIS PAS UNE BETE EN FORTH JE ME PROPOSE D'ECRIRE UNE DEMONSTRATION TOURNANTE DE TURBO FORTH. ELLE CONSITERAIT A EXPLIQUER L'INTERET DE CE LANGAGE ET LA QUALITE DE TUR-FTH. MA DISPO ME PERMETTRAIT D'ARRIVER A UN BON RESULTAT SOUS 8-9 SEM.
COMME JE NE PEUT ME SUBSITUER AUX AUTEUR DE CET OUTIL ET QUE QUELQUES IDEES ME MANQUENT ENCORE JE COMPTE SUR VOUS TOUS POUR DES TONNES DE SUGGESTIONS.
MON DESIR PROFOND EST DE FAIRE DE FORTH UN LANGAGE DE + EN + DIFFUSEE. ECRIVER SUR LE FORUM OU DIRECTEMENT DANS MA BAL 'TICK'... BALTICK AHAHAHAH (Ndlr: voici de l'humour froid)

SECRETAIRE 13.10.88 08h38
pour rappel, les concepteurs de TURBO-FORTH 83-Standard sont:
- Marc PETREMANN (moi-meme)
- Michel ZUPAN (FORTH7)

Mais depuis sa sortie officielle, ce langage est maitrise aussi par d'autres cracks que je ne citerai pas ici. Donc, excellente initiative de TICK. Mais il y a mieux qu'une simple demo tournante a realiser, ce serait un veritable TUTORIAL. J'avais deja commence qque chose en ce sens, mais par manque de temps... no comment.

En ce qui concerne le manuel de TURBO-Forth, nous ne pensons pas l'achever avant la fin du premier trimestre 89.
Toujours ce manuel TF83: nous sommes revenus sur la presentation initalement envisagee et avons decide de le decouper en trois parties distinctes:
1) prise en main: pour ceux ne connaissant absolument rien a FORTH.
2) glossaire detaille et thematique: les mots sont expliques par ordre logique; ex: IF ELSE THEN CASE OF ENDOF ENDCASE... etc...
3) programmation avancee: des themes de programmation tres ardus. Pour bien demontrer que FORTH n'est pas de la petite biere...

Donc, fideles adherents programmeurs Forthiens, nous comprenons votre impatience, mais nous souhaitons repondre au mieux de votre interet a votre soif de connaissance sur FORTH.

Ah, petite disgression: ayant rendu visite a F32, nous avons teste TURBO-Forth sur station de travail APOLLO (equipee je crois d'un 68000 qquechose) et... ca marche!!! (en mode emulation IBM, precisons-le).

Petit message a DUMUR: quand mettons-nous au point un emulateur 68000 sur PC?

TICK 13.10.88 10h25
CHER SECRETAIRE, JE ME PROPOSE POUR UN PETIT COUP DE MAIN

(suite page 23)

Fred Behringer
Straßbergerstr. 9c/519
D-8000 München 40
Germany

FORTH ON THE IBM : PRIME NUMBERS

There is a well known public domain version of FORTH83 for the IBM-PC by Laxen and Perry, which is screen-based, i.e., it makes use of FORTH's block concept.

There is a very active group of FORTH enthusiasts residing in Paris:

ASSOCIATION JEDI
17, rue de la Lancette
F-75012 PARIS .

Recently, JEDI activist Marc Petremann and his colleagues have developed a FORTH83 version based on the MS-DOS file concept. They call it Turbo-FORTH. It is simply marvellous and I have started to convert their French system into a German system.

I understand that they are going to produce an English version as well as a Spanish version in addition to the French version they started with.

I may honestly say that their's is exactly the DOS-based system I was waiting for. I wouldn't care too much about portability. I want to use it on my machine and want to use it now and I want to be able to talk about my doings to others having similar machines.

Marc Petremann and his colleagues have also written a book on Laxen and Perry's FORTH83 (in French) which covers most of Turbo-FORTH. A manual on Turbo-FORTH will soon be available.

The JEDI people have not placed their Turbo-FORTH into public domain. However, the prices they ask (FF 37.- for the main disk, FF 100.- for system plus tools [full screen editor, floating point package etc.]) are very low indeed.

As to my own efforts: I have completed a translation into German of an 8-page short description (LISEZ.MOI ==> LIES.DAS) and the preparation of a German version of the main system. The translation of an inline HELP file, explaining the FORTH words by evoking HELP <word>, is nearly completed.

The three disks distributed by JEDI contain a piece of FORTH called ERATHOS, a program for determining the prime numbers up to n by the famous sieve method of Eratosthenes: For each i between 1 and $\sqrt{n}$, mark all multiples of i. The numbers remaining unmarked are the prime numbers.

JEDI's program is ment to be a benchmark for demonstrating the speed FORTH is able to develop: The primes between 1 and 10000 in 2.14 seconds on my AT clone.

JEDI's program is a ready-to-go program, not taking advantage of the fact that it suffices to consider only those multiples of i which do not exceed $\sqrt{n}$. I was wondering how much time could be saved by adding some sort of a square root restriction. What was needed was a quick-and-dirty method for determining the square root of an unsigned 16-bit number u.

I tried two SQR programs which I found in the book "FORTH 83" by Zech. They apply Newton's method or the method of quadratic extension. Then I constructed one of my own: It applies bisection (interval halving).(The method of bisection for determining an argument x where the function involved assumes a prescribed value (here $f(x) := x^2 := u$) works with any continuous function on a compact interval. This is simply an application of the intermediate value theorem.)

The time needed for determining the square root of a given number can be neglected if compared with the time needed for calculating the first 1000, say, prime numbers. So, any square root program, as brute as it may be, will serve the purpose. On the other hand, the time needed for calculating the prime numbers by JEDI's program could be reduced by one third if a square root mechanism was inserted.

Let us begin by reproducing JEDI's program (kind permission taken for granted).

```
\ **********************
\ Sieve of Eratosthenes
\ **********************

FORTH DEFINITIONS DECIMAL
8190 CONSTANT SIZE \ Pseudo constant, size of table from 1 to n/2

: PRIMES ( n -- )  \  display number of prime numbers from 1 to n
          4 MAX DUP 2/ 1- IS SIZE PAD SIZE 1 FILL
          2 SIZE 0
          DO   PAD I + C@
               IF  I 2* 3 + DUP I +
                   BEGIN  DUP SIZE <
                   WHILE  0 OVER PAD + C! OVER +
                   REPEAT DROP DROP 1+
               THEN
          LOOP
          .  ." Primes between 1 and " . ;

: LISTING ( -- ) \ displays list of prime numbers determined by PRIMES
     CR 1 . 2 . SIZE 0 DO PAD I + C@ IF I 2* 3 + . THEN STOP? ?LEAVE LOOP ;
```

Now load the Turbo-FORTH program TIMER by INCLUDE TIMER. Then determine the time needed for the primes between 1 and 10000 by

```
1 CHRONO" 10000 PRIMES"
```

As a result, I obtained 2.14 sec on my AT.

The next step was to include a square root program SQR into the above. So, here is my modified version of JEDI's program:

```
\ ***************************
\ Sieve of Eratosthenes - II
\ ***************************

echo off
FORTH DEFINITIONS DECIMAL                    \ definitions in vocabulary FORTH
8190 CONSTANT SIZE                           \ pseudo constant,
                                             \ size of table from 1 to u
: PRIMES ( u -- )                            \ displays number of
                                             \ prime numbers from 1 to u
   DUP SQR 2/ PAD 2- ! .                     \ store √u/2 in PAD-2
   4 MAX                                     \ minimum size
   DUP 2/ 1- IS SIZE                         \ size of table from PAD
   PAD SIZE 1 FILL                           \ fill table with 1's
   2 PAD 2- @ 0                              \ for I=0 to √u/2 do the following
   DO   PAD I + C@                           \ if PAD+I contains 1, then do:
        IF  I 2* 3 + DUP I +                 \ multiples of 2*I+3
            BEGIN  DUP SIZE <                \ while not yet table end
            WHILE  0 OVER PAD + C! OVER +    \ store 0 in PAD+2*I+3
            REPEAT DROP DROP                 \ repeat for next multiple
        THEN                                 \ if PAD+I = 0, then skip
   LOOP                                      \ repeat if not yet I = √u/2
   SIZE 0                                    \ for I=0 to u/2 do the following
   DO   PAD I + C@                           \ if PAD+I = 1, then
        IF    1+                             \ increment counter
        THEN
   LOOP
   . ." Primes between 1 and " . ;          \ display number of primes

: LISTING ( -- ) \ displays list of prime numbers determined by PRIMES
     CR 1 . 2 . SIZE 0 DO PAD I + C@ IF I 2* 3 + . THEN STOP? ?LEAVE LOOP ;
```

As to the square root program SQR, I first tested the following bisection program of my own invention:

```
: SQR  ( u1--u2 )                   \ Greatest number not greater than √u1
  HERE ! 0 PAD !                    \ Save u1, initialize PAD (accumulator)
  1 2 4 8 16 32 64 128              \ Keep powers of 2
  8 0                               \ Repeat 8 times
      DO DUP PAD @ + DUP * HERE @   \ Tentatively add next power of 2 to PAD
         u> IF DROP 0 THEN PAD +!   \ PAD² > u1 ?  Yes: drop it. No: do it
      LOOP                          \ Repeat the above
  PAD @ ;                           \ Collected powers of 2 represent INT(√u1)
```

The modified sieve program containing this square root program yielded the prime numbers between 1 and 10000 in 1.65 seconds (as compared to 2.14 sec

with the unmodified French version).

To measure the time performance of SQR itself, I typed in 1 CHRONO" TEST" with

```
: TEST
  65535 0 DO I SQR DROP LOOP ;      \ Execute SQR for 0 to 65535
```

The result was 155.39 sec, which gives a crude avarage of 2 msec for one run.

This done, I was wondering about the performance of the two square root programs from the book by Zech. The first program reads:

```
: SQR ( u--u )                      \ Newton's iteration
  DUP 0= IF EXIT THEN
  16
  BEGIN
        2DUP
        0 SWAP UM/MOD SWAP DROP
        OVER +
        2/
        DUP ROT
        -
        ABS
        2/
   WHILE
   REPEAT ;
```

The above CHRONO test yielded 56.03 sec, which is only one third of the value for the method of bisection.

The second square root program which I found in Zech's book is in machine code. It reads:

```
CODE SQR ( u--u )   CX POP   08 # DI MOV   0 # AX MOV   AX DX MOV
     BEGIN STC
           DX RCL
           CX SHL   AX RCL
           CX SHL   AX RCL
           DX AX CMP
           >=
           IF   DX AX SUB
                DX INC
           ELSE DX DEC
           THEN
           DI DEC
     0= UNTIL
     DX SAR
     DX PUSH  NEXT END-CODE
```

This works by the method of quadratic extension. The above CHRONO test yielded 7.74 sec, which shows that machine code programming is still unbeatable. However, it also demonstrates how easy it is to incorporate pieces of machine code into FORTH.

The above modification of JEDI's prime number program PRIMES only works correctly if u < 32767. This is because MAX does not work with unsigned numbers. Here is my suggestion for UMAX, a modified MAX:

```
: UMAX ( u1,u2--u3 )      \  u3 = max(u1,u2), all numbers unsigned
      2DUP U<             \  u1 < u2 ?
      IF SWAP THEN        \  yes (no), then prepare u1 (u2) to be dropped
      DROP ;              \  drop it and keep the remaining
```

A similar arrangement could be made for UMIN:

```
: UMIN ( u1,u2--u3 )      \  u3 = min(u1,u2), all numbers unsigned
      2DUP U> IF SWAP THEN DROP ;
```

Before making further modifications, let us briefly discuss a few items:

(1) The maximal data stack length,multiplied by 2, determines the maximal range of numbers which can be searched for primes. So, let us insert an error message and an exit switch for cases of violation.

(2) The range of numbers to be searched for primes may exceed 32767. Hence, "."

in the message part must be replaced by "U." .

(3) LIST in JEDI's program stops whenever a key is pressed,and continues thereupon when any key is touched, or exits when the key then touched is CR. If we envisage some 6000 prime numbers to be displayed on the screen, it will certainly not be bad to be able to prescribe the number the list is to start with. A sieve program thus modified could read as follows:

```
\ ****************************
\ Sieve of Eratosthenes - III
\ ****************************

echo off
FORTH DEFINITIONS DECIMAL
8190 CONSTANT SIZE
     VARIABLE SQROOT
: PRIMES ( u -- )
   DUP U2/ SP@ 100 - PAD - U>               \ enough space ?
        IF ABORT" Not enough space "         \ if not, then print message
        THEN                                 \ and abort, otherwise
   DUP SQR 2/ SQROOT !                       \ store √u/2 in SQROOT
   4 UMAX                                    \ MAX replaced by UMAX
   DUP U2/ 1- IS SIZE                        \ 2/ replaced by U2/
   PAD SIZE 1 FILL                           \ fill table with 1's
   2 SQROOT @ 0                              \ for I=0 to √u/2 do the following
   ?DO  PAD I + C@                           \ if PAD+I contains 1, then do:
        IF  I 2* 3 + DUP I +                 \ multiples of 2*I+3
            BEGIN  DUP SIZE U<               \ while not yet table end
            WHILE  0 OVER PAD + C! OVER +    \ store 0 in PAD+2*I+3
            REPEAT DROP DROP                 \ repeat for next multiple
        THEN                                 \ if PAD+I = 0, then skip
   LOOP                                      \ repeat if not yet I = √u/2
   SIZE 0                                    \ for I=0 to u/2 do the following
   DO   PAD I + C@                           \ if PAD+I = 1, then
        IF    1+                             \ increment counter
        THEN
   LOOP
   CR .
   ." prime numbers between 1 and " U. ;     \ display number of primes

: LISTING ( u -- )          \ displays list of prime numbers determined by PRIMES
         CR DUP 4 <
         IF
         DROP 0 1 . 2 .
         ELSE
         3 -                                 \ display first two primes
         THEN
         U2/
         DUP SIZE 1- >
         IF ABORT" Try again ! "
         THEN
         SIZE SWAP                           \ for I=0 to u/2
         DO                                  \ do the following
            PAD I + C@                       \ if PAD+I = 1, then
            IF I 2* 3 + U.
            THEN
            STOP? ?LEAVE
         LOOP ;
```

The CHRONO test, done with the above ERATHOS-III version, yielded 1.65 sec for 10000 numbers searched, the same as with ERATHOS-II. Hence, the modifications applied to ERATHOS-II did not slow down the time of performance. The time needed for determining all prime numbers between 1 and 65535 was 11.26 sec for ERATHOS-III.

Hitherto, we have made use of the fact that only odd numbers can be prime. In order to be able to employ as much RAM space as possible, from a total of 64K, addressable by means of 16-bit numbers only, we are now going to also exclude numbers which can be divided by 3. In other words, we wish to skip over all numbers devidable by 2 and/or 3.It is clear that the overall saving will amount to 2/3: Every second number is devidable by 2, which yields a reduction of the set of numbers to be searched by 1/2 or 3/6. Every third number is devidable by 3, which yields a reduction by 1/3 or 2/6. Every sixth number is dividable by 2 and 3 and has thus been counted twice in the sum 3/6 + 2/6. Hence, the set of numbers to be searched reduces to 6/6 - 3/6 - 2/6 + 1/6, or 1/3.

Let us pause for a moment and consider the following scheme. (For reasons which are obvious, we start with ud = 5.)

```
ud     5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31
I      0   |   1   |   |   |   2   |   3   |   |   |   4   |   5   |   |   |   6   |   7   |   |   |   8   |   9

ud    32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56
I      |   |   |  10   |  11   |   |   |  12   |  13   |   |   |  14   |  15   |   |   |  16   |  17   |

ud    57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81
I      |   |  18   |  19   |   |   |  20   |  21   |   |   |  22   |  23   |   |   |  24   |  25   |   |

ud    82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104
I      |  26   |  27   |   |   |  28   |  29   |   |   |  30   |  31   |   |   |  32   |  33   |

ud   105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123
I      |   |  34   |  35   |   |   |  36   |  37   |   |   |  38   |  39   |   |
```

Here, ud denotes the generic member of the set of all numbers to be searched for primes, whereas I, the loop index in the program to follow, corresponds to the members of the set of numbers remaining when the numbers which can be divided by 2 and/or 3 are deleted.

Looking at the scheme above,we see that every even I has skipped over 2*I positions, whereas for odd I's this number is 2*I-1. Since ud, the number corresponding to the loop index I, is I plus the number of positions skipped, and since we start with 5, we have

```
ud = 3*I+5 for I even
ud = 3*I+4 for I odd
```

This is the relation we need for the final recovery of the prime numbers from the condensed table. Now let us briefly discuss how to represent, for each I with its associated ud , those I's which are associated with multiples of ud. Let us take a purely experimental approach, using the scheme above:

For ud = 5, the next multiple of ud corresponding to an existing I from the set of loop indices, is ud = 5*5 . The next again is ud = 5*7, followed by 5*11, 5*13, 5*17, 5*19, 5*23, 5*25 ..., etc.

Similarly, for ud = 7, we have 7*5, 7*7, 7*11, 7*13, 7*17, 7*19, 7*23, 7*25,... In general, ud*5, ud*7, ud*11, ud*13, ...

Back to I, this amounts to I = 7, 10, 17, 20, 27, ... for ud = 5 (I = 0). And I = 10, 15, 24, 29, 38, ... for ud = 7 (I = 1). Again, I = 17, 24, 39, 48, ... for ud = 11 (I = 2). And, I = 20, 29, 46, 55, ... for ud = 13 (I = 3), etc.

It is easy to see that this will be covered by the following relations:

```
I
+ 4*I+7      for I even
+ 2*I+3      for I even
+ 4*I+5      for I odd
+ 2*I+3      for I odd
```

In other words, for any I we have that

I + 4*I+7 + 2*I+3 + 4*I+5 + 2*I+3 + 4*I+7 + 2*I+3 + ...

represents a multiple of the corresponding ud (which cannot be a prime number). The consecutive first terms of the form shown above will therefore have to produce a 0 in our condensed table. Here is the complete program: (Note that in quite a few places we had to replace a 16-bit unsigned number [or a 16-bit operation] by its 32-bit counterpart.)

```
: UD< ( ud1,ud2--fl )          \ fl=TRUE if ud1 < ud2 ; fl=FALSE otherwise
      DUP 3 PICK =             \ upper parts equal ?
      IF                       \ yes, then
      ROT 2DROP U<             \ lower parts are decisive
      ELSE                     \ no, then
         SWAP DROP ROT DROP    \ upper parts
         U<                    \ are decisive
      THEN ;

: UD> ( ud1,ud2--fl )          \ fl=TRUE if ud1 > ud2 ; fl=FALSE otherwise
      2SWAP UD< ;

: UD= ( ud1,ud2--fl )          \ fl=TRUE if ud1 = ud2 ; fl=FALSE otherwise
      D- D0= ;

: UDMAX ( ud1,ud2--ud3 )       \ ud3 = max(ud1,ud2), unsigned, double precision
        4DUP UD<               \ ud1 < ud2 ?
        IF 2SWAP               \ yes (no), then prepare ud1 (ud2) to be dropped
        THEN
        2DROP ;                \ drop it and keep the remaining
```

```
: SQR ( ud1--ud2 )                  \ greatest number not greater than √ud1
  HERE 2! 0 PAD !                   \ save ud1, initialise PAD (accumulator)
  1 2 4 8 16 32 64 128 256 512      \ keep
  1024 2048 4096 8192 16384 32768   \ powers of 2
  16 0                              \ repeat 16 times
    DO DUP PAD @ + DUP UM* HERE 2@  \ tentatively add next power of 2 to PAD
         UD> IF DROP 0 THEN PAD +!  \ PAD² > ud1 ?  Yes: drop it. No: do it
      LOOP                          \ repeat the above
  PAD @ ;                           \ collected powers of 2 represent INT(√ud1)


: NOT! ( addr-- )                   \ logical conversion of content of
  DUP @ NOT SWAP ! ;                \ variable in address addr


\ ***************************
\ Sieve of Eratosthenes - IV
\ ***************************

echo off
FORTH DEFINITIONS DECIMAL
8190 CONSTANT SIZE                          \ table size
     VARIABLE SQROOT                        \ (√ud)/3
     VARIABLE FLAG                          \ adjust for table entry odd or even
: PRIMES ( ud -- )

   5. UDMAX                                 \ min for start is 5
   2DUP 5. D- 3 UM/MOD IS SIZE DROP         \ table size one third
   SIZE SP@ 100 - PAD - U>                  \ enough space ?
        IF ABORT" Not enough space "        \ if not, then print message
        THEN                                \ and abort, otherwise
   2DUP SQR 3 / 1+ SQROOT !                 \ store (√u)/3 in SQROOT
   PAD SIZE 1 FILL                          \ fill table with 1's
   FLAG ON                                  \ initialize FLAG (-1,0,-1,0,...)
   SQROOT @ 1+ 0                            \ for I=0 to (√ud)/3 do the following
   ?DO  FLAG NOT!                           \ -1 if I odd, 0 otherwise
        I                                   \ next candidate
        PAD I + C@                          \ if PAD+I contains 1,
        IF                                  \ then do:
          BEGIN
            I 2* 2* 7 + FLAG @ 2* + +       \ add 4*I+7(6)
            DUP SIZE U> NOT                 \ if table end, then leave
            IF DUP PAD + 0 SWAP C! THEN     \ store 0 in PAD+4*I+7(6)
            I 2* 3 + +                      \ add 2*I+3
            DUP SIZE U<=                    \ if table end, then leave
            WHILE DUP PAD + 0 SWAP C!       \ store 0 in PAD+2*I+3
          REPEAT                            \ repeat for next pair of multiples
        THEN DROP                           \ if PAD+I = 0, then skip
   LOOP                                     \ repeat if not yet I = (√ud)/3
   3                                        \ initialize counter

   SIZE 1+ 0                                \ for I=0 to SIZE do the following
   DO   PAD I + C@                          \ if PAD+I = 1, then
        IF 1+ THEN                          \ increment counter
   LOOP
   CR . ." prime numbers between 1 and "    \ display
   UD. ;                                    \ number of primes
```

The CHRONO test mentioned above yielded 1.15 sec for 10000 numbers searched for primes. In other words, in spite of the fact that we had to include double precision operations, the saving gained is not only with space but with time, too.

A search from 1 to 65535 was done in 7.96 sec. The maximal range, available on my machine with the full Turbo-FORTH system installed, is 111600, which yields a total of 10585 prime numbers in 13.85 sec.

(Note that ud must be typed in as a double precision number. If, for instance, we want to search for prime numbers in the range of 0 to 102999, the FORTH command should read "102999. PRIMES" .)

The following FORTH word LISTPRIMES is for displaying the prime numbers in the range determined by xxx. PRIMES . By entering "xxx. STARTPRIME 2!" the starting number for the numbers to be displayed can be changed. The process of displaying prime numbers can be interrupted any time by hitting any key, and resumed by again hitting any key except <CR>. If the second key pressed is <CR>, then the process of displaying will be interrupted definitely.

```
2VARIABLE STARTPRIME                        \ starting number for list search
          0. STARTPRIME 2!                  \ is 0 by default
                                            \ can be altered to xxx.
                                            \ by XXX. STARTPRIME 2!
: LISTPRIMES ( -- )   \ displays list of prime numbers determined by PRIMES
          STARTPRIME 2@ 5. D-               \ starting number for table search
          6 UM/MOD SWAP DROP 2*             \ adjusted to even table entry
          DUP 0 U< IF DROP 0 THEN           \ adjust to 0 if (STARTPRIME) < 5
          DUP SIZE U>                       \ if greater than table size,
          IF ABORT" Try again ! " THEN      \ then abort
          CR ." Prime numbers between "     \ display opening message
          DUP 0=                            \ if 5 or less,
          IF   ." 0 "                       \ then start with 0
          ELSE DUP 3 UM* 5. D+ UD.
          THEN                              \ display range
          ." and "                          \ of numbers
          SIZE 3 UM* 5. D+ UD. ." :" CR     \ searched
          DUP 0=                            \ if table entry smaller than 0,
          IF 1 . 2 . 3 . THEN               \ display first 3 primes
          FLAG ON                           \ initialize FLAG
          SIZE 1+ SWAP                      \ for I=INT(STARTPRIME/6)*2 to SIZE
          DO                                \ do the following
           FLAG NOT!                        \ -1 if I odd, 0 otherwise
           PAD I + C@                       \ if PAD+I = 1,
```

```
        IF I 3 UM* 5. D+ FLAG @          \ then display
           DUP D+ UD.                    \ corresponding
        THEN                             \ prime number
        STOP? ?LEAVE                     \ if any key pressed, then stop;
       LOOP ;                            \ if thereafter CR, then exit;
                                         \ otherwise loop back
```

Finally, here is a FORTH word for displaying all pairs of prime numbers of distance 2u. Obviously, there is no need for u being double precision. The usual notion of a "pair" of prime numbers is covered by u=1 . The process of displaying pairs of prime numbers can be switched on and off by entering "DISPLAY ON" or "DISPLAY OFF" . If DISPLAY OFF, then only the number of pairs is output. If DISPLAY ON,the process can be interrupted by the same means as with LISTPRIMES.

```
VARIABLE DISPLAY                  \ ON/OFF
VARIABLE DISTANCE                 \ distance between primes in pair
VARIABLE DISTEMP                  \ every third table entry skipped
VARIABLE DISFLAG                  \ add table entry skipped ?
VARIABLE PAIRCOUNT                \ number of pairs
VARIABLE PAIRSTART                \ start of current pair
VARIABLE PAIRPOINTER              \ where in current pair ?
VARIABLE PAIRFLAG                 \ add table entry skipped ?

: PAIRS ( u-- )                   \ displays number of pairs of distance 2u
                                  \ of prime numbers determined by PRIMES
 CR DUP DISTANCE !                   \ keep distance
 PAIRSTART OFF PAIRCOUNT OFF         \ initialization
 PAIRFLAG ON
 1 =                                 \ if DISTANCE = 1,
 IF PAIRCOUNT 1+!                    \ then increase PAIRCOUNT
    DISPLAY @                        \ and if display on,
    IF 3 . 5 . SPACE                 \ then display pair (3,5)
       ASCII - EMIT SPACE            \ and spaces
    THEN
 THEN
 DISPLAY @ 0=                        \ if DISPLAY switched off,
 IF ." Wait a few seconds !" THEN    \ then display message
 BEGIN                               \ main loop <
  PAIRFLAG NOT!                      \ if position of 2nd prime in
  PAIRSTART @ DISTANCE @ +           \ current pair is greater than
  DUP 0 3 UM/MOD SWAP DROP - SIZE U< \ table size, then exit
 WHILE                               \ otherwise
   PAIRPOINTER OFF                   \ initialize PAIRPOINTER
   PAIRFLAG @ NOT DISFLAG !          \ and local entities
   DISTANCE @ DISTEMP !              \ DISFLAG and DISTEMP
    PAD PAIRSTART @ + C@             \ if nonprime, then skip
    IF                               \
     PAIRSTART @ PAIRPOINTER @ +     \
     SIZE U<=                        \ if greater than table size,
     IF                              \ then skip
      BEGIN                          \ otherwise loop <
       PAIRPOINTER 1+!               \ increase pointer
       DISFLAG NOT!                  \
       DISFLAG @ DISTEMP +!          \
       PAIRPOINTER @ DUP PAIRSTART @ + \ if table entry <> 0 or
       PAD + C@ 0=                   \ pair limit exceeded,
       SWAP DISTEMP @ U< AND         \ then exit
      WHILE                          \
      REPEAT                         \ <
       PAIRPOINTER @ DUP PAIRSTART @ + \ if table entry = 0 or
       PAD + C@ 1 =                  \ pointer <> distance,
       SWAP DISTEMP @ = AND          \ then skip
       IF                            \
        DISPLAY @ 0<> STOP? NOT AND  \ if DISPLAY switched off,
        IF                           \ then skip
         PAIRSTART @ 3 UM* 5. D+     \ display 1st prime
         PAIRFLAG @ 0 D+ 2DUP UD.    \ odd or even ?
         DISTANCE @ 0 D2* D+ UD.     \ display 2nd in pair
         SPACE ASCII - EMIT SPACE    \ space between pairs
        ELSE                         \
         DISPLAY @                   \ if STOP and CR, <
         ABORT" Display aborted !  OK" \ then abort
        THEN                         \
        PAIRCOUNT 1+!                \ increase paircounter <
       THEN                          \
     THEN                            \
    THEN                             \ increase current pairstart <
  PAIRSTART 1+!                      \
 REPEAT                              \ <
 CR PAIRCOUNT @ U.                   \ display number of pairs
 ." prime pair(s) of distance "      \ of distance 2u
 DISTANCE @ 0 D2* UD. ;
```

The maximal range available on my IBM AT clone is from 1 to 110000 or so. The maximal distance two prime numbers can be apart in this range seems to be 72, which is assumed by the pair 31397 31469 .

Use of a Forth-Based Prolog for Real-Time Expert Systems

II. A Full Prolog Interpreter Embedded in Forth

L. L. Odette

Applied Expert Systems, Inc.
5 Cambridge Center
Cambridge, MA 02142

W. H. Paloski

KRUG International
Technology Life Sciences Division
17625 El Camino Real, Suite 311
Houston, TX 77058

Abstract

In this article we outline the design of a Prolog interpreter embedded in Forth. The interpreter is the basis of the expert system component of an astronaut interface for a series of Spacelab experiments. The expert system is described in Part I of this article [PAL87]. Here we describe our approach to the representational issues in designing the programming machinery needed to interpret Prolog programs: (1) the internal representation of Prolog objects and (2) the representation of the state of a Prolog computation. We also describe the Forth-Prolog interface we use to support the mixed language programming that is necessary to handle the real-time data acquisition and control tasks involved in the application.

Our goal is to combine the advantages of Forth for real-time programming and the advantages of Prolog for symbolic reasoning. To take advantage of the large body of Prolog code we have developed for previous applications, we implemented the "core" Prolog system described in [CLO81] that is compatible with the widely available implementations.

Introduction

The text that follows briefly describes the implementation of the Prolog interpreter used in our application [PAL87]. The interpreter is fully Clocksin and Mellish compatible, using the standard Edinburgh syntax and providing the majority of the built-in predicates described in [CLO81] (some file I/O predicates are not implemented); however, it's a "tiny" Prolog in that it can fit in the 64K of the small model Forth. It is particularly suitable for Prolog applications that can leverage off the underlying Forth system, such as the knowledge-based control system described in [PAL87]. The full source code for the interpreter (about 100 screens) is available from the Forth Interest Group as volume 5 of the Forth Model Library.

The intent here is to cover briefly the main issues involved in implementing a Prolog interpreter in the context of one particular implementation in Forth. For this reason familiarity with Prolog is assumed. The first section describes the Forth data structures used to represent Prolog terms. The next three sections address memory allocation, how variables are bound and how Prolog procedures are invoked. The fourth section describes the implementation of built-in predicates and the interface between Prolog and Forth. The final section is a list of the built-in predicates that have been implemented.

Representation of Terms

Syntactically, there are two types of objects in Prolog – simple and complex objects. Constants and variables have no syntactic structure and are examples of simple (unstructured) objects. On the other hand, lists and predications have structure, and these are complex objects. Semantically, constants and variables are quite different from each other, and they need to have different internal representations. Moreover, for the sake of efficiency, internal representations of Prolog objects may incorporate other type distinctions (e.g., names and numbers may have separate internal representations even though they are both of "constant" type).

The Prolog interpreter needs to be able to identify the type of a Prolog object. Common schemes for typing include using separate memory areas for different object types or incorporating a type (tag) field into an object pointer. The former is probably not a good choice for a straightforward Forth implementation where it is natural to have the names of objects and the objects themselves intermingle in the dictionary. The latter was not compatible with the use of 16-bit pointers in this implementation. Instead, the typing scheme used here has the interpreter get the type of an object not from the value of the pointer or the pointer itself, but from the object pointed to. The following text describes this scheme in detail.

Bill Paloski has been building data acquisition and patient monitoring systems since 1977. Before completing his doctorate in biomedical engineering at Rensselaer Polytechnic Institute in 1982, he spent one year at the Oak Ridge National Laboratory and four years at the S. R. Powers Trauma Research Center in Albany, New York. After that he spent three years as an assistant professor at Boston University and then moved to his current position with KRUG International at the Johnson Space Center. In addition to real-time computing, his research interests are in pulmonary physiology, critical care monitoring, and physiological adaptation to weightlessness.

Dr. Odette is international technology marketing manager at Applied Expert Systems, Inc. While at Applied Expert Systems, Dr. Odette has played a major role in designing and implementing the Apex development environment. His work has concentrated on computer language and compiler design. He has also worked on a wide range of technology marketing and delivery issues. Prior to joining Apex, Dr. Odette was a principal of Telphi Systems, Inc., a manufacturer of communications equipment. He was responsible for the design and implementation of data base management systems. Before founding Telphi, Dr. Odette did research at the Massachusetts Institute of Technology where he received a Ph.D. in electrical engineering. He did early work in neural network modeling, where he developed advanced circuit models of neurons. His thesis work focused on perception in the visual system.

Al Krever majored in theater at Emerson College, Boston, Massachusetts, in the mid-1960s but then became interested in special purpose programming. After spending time with DEC and Honeywell in the late 1960s and throughout the 1970s, he joined FORTH, Inc., in 1980. Since then he has been designing custom applications and developing an international reputation as a Forth educator. His current interests are in applying AI to real-time monitoring and process control.

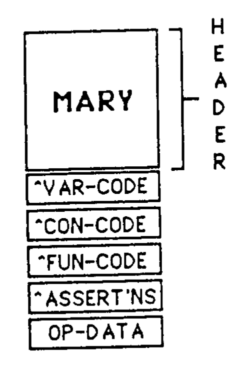
Allison West received a B.S. in biomedical engineering from the University of Iowa in 1981 and an M.S. in electrical science from the University of Michigan in 1984. She currently works for KRUG International, a NASA contractor, in Houston, Texas. Her interests include real-time control, data acquisition systems, and expert systems.

Primitive Terms: Constants, Variables and Numbers

Constants and variables have name and link fields combined to form a header just like any Forth word, and these are the only Prolog objects with name fields (referred to as named objects). Immediately following the header are an additional 5 fields comprising 3 vectors and 2 data fields. In order of increasing memory address the fields are:

Name	Contents	Use (object pointer/goal)
variable-code field	0	object = variable/(not used).
constant-code field	The cfa of the Forth word `RESOLVE.SINGLE`.	object = constant/ call function (no args).
function-code field	The cfa of the Forth word `RESOLVE.FUN`.	object = functor/ call function.
assertion field	A pointer to the list of assertions for this functor.	
functor data field	Functor data on precedence, position, associativity.	

For example, the representation of the constant term **mary** looks like:

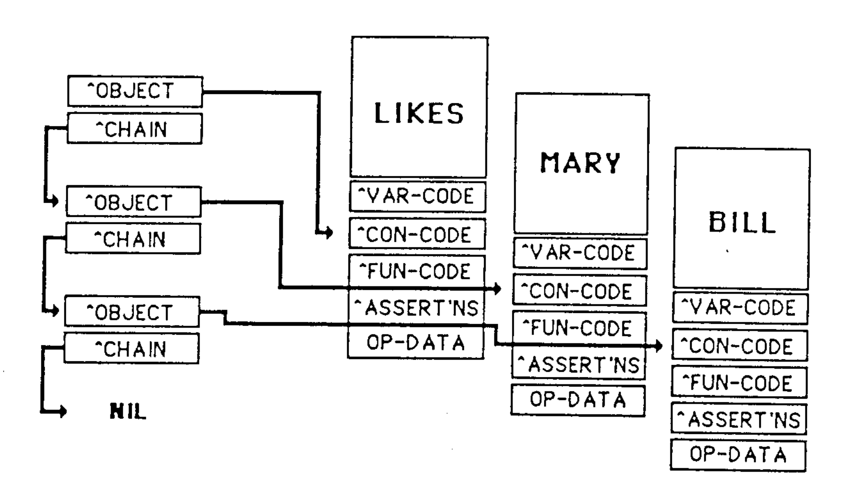


The vector fields of a named object have a dual use: they are used to transfer control at run time when a named object is to be executed as a goal or is the main functor of a goal, and they also are used by the unifier to identify the type of an object. Typing works as follows. If a pointer to a named object points to the variable-code field, then the object is treated as a logical variable. If the pointer points to the constant-code field, the object is treated as a constant. If the first element of a complex term points to the function-code field of some named object, then the complex term is interpreted as a function (predication or procedure) whose functor is the first element and whose arguments are the remaining elements of the complex term.

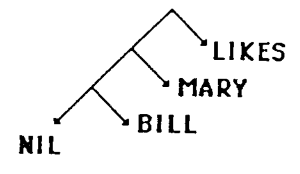
Numbers are the one exception to the typing scheme. The fact that an object is a number is determined either directly from the pointer to it or from the memory location the pointer references. In the first case, if an object pointer points to an address less than 256, then the pointer is interpreted as the object, that is, a number. This approach allows a more compact representation of small numbers at the expense of run-time checking. In the second case, if the first memory word of the object points to the next memory word, then the contents of the next word is interpreted as the number.

Complex Terms

Complex terms are represented as chains of word (memory location) pairs. The first word of a pair points to an object; the second word of a pair points to the rest of the chain (for LISPers: all complex terms are lists—the word pairs constitute a CONS cell). A chain is terminated when the second word of a pair points to an object that is not an object chain. For example, lists are terminated with a pointer to the special object **NIL**, which is both a constant and the representation of the empty list. The list of terms **[likes,marry,bill]** is represented internally as:

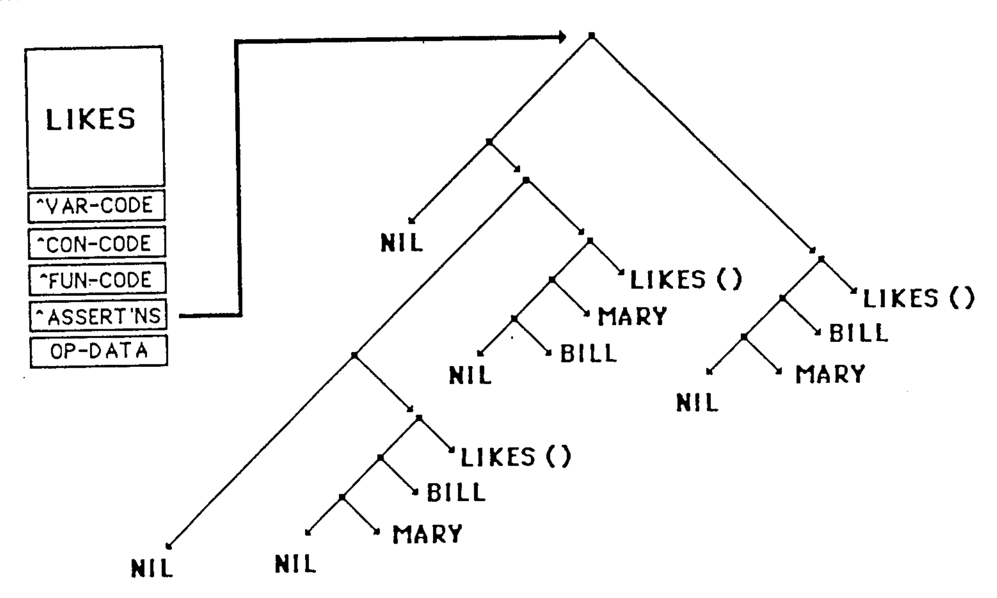


A more concise representation of the structure of complex terms is a tree diagram whose nodes represent word pairs. Each right subtree of a node is a representation of the object pointed to by the first word of the pair. Each left subtree of a node is a representation of the object pointed to by the second word of the pair (i.e., the rest of the chain). In this notation, the tree representation of the list **[likes,mary,bill]** is:



[(likes(bill,mary)), (likes(mary,bill) :- likes(bill,mary))]

is stored as:



Memory Usage

This implementation intermingles the object-name and program spaces in the Forth dictionary and provides for no garbage collection of either names or program. A heap might be used to handle de-allocation of program space following the retraction of a clause, although this procedure would need to be handled with care because the retracted clause may still have a reference—as an untried alternative that may be required to resatisfy an earlier goal.

In addition to the name and program spaces and the space allocated to the usual Forth return and data stacks, there are a stack that holds variable bindings (binding stack) and a separate stack for control information (goal stack). These stacks are made as large as possible because this is where the action is in a Prolog computation. In the version used in our application [PAL87] 32K bytes are allocated for each stack.

The goal stack is used for saving goal frames during a computation (it corresponds to the return stack in Forth). Each goal frame has 6 fields (2 bytes each) that will be described later. One of these fields points into the binding stack and indicates the top of the binding stack relative to that goal frame (i.e., the set of bindings in effect when that goal frame was entered). In this way variable bindings can be redone, if backtracking is necessary, simply by resetting the binding stack pointer. A new goal frame is allocated on successful resolution of a goal with the head of an assertion by advancing the goal stack pointer, copying the contents of the last goal frame and updating the fields.

Variable Bindings

Variables are bound as a result of unification during the resolution step. Each use of a variable name must be associated somehow with the call to the procedure that the variable appears in. We use goal frame indices for this purpose. Prior to unification, a pointer to the goal is associated with the value of the current index (`INDEX.C`) and a pointer to the head of the first clause in the assertions list is associated with the value of the next index (`INDEX.N`).

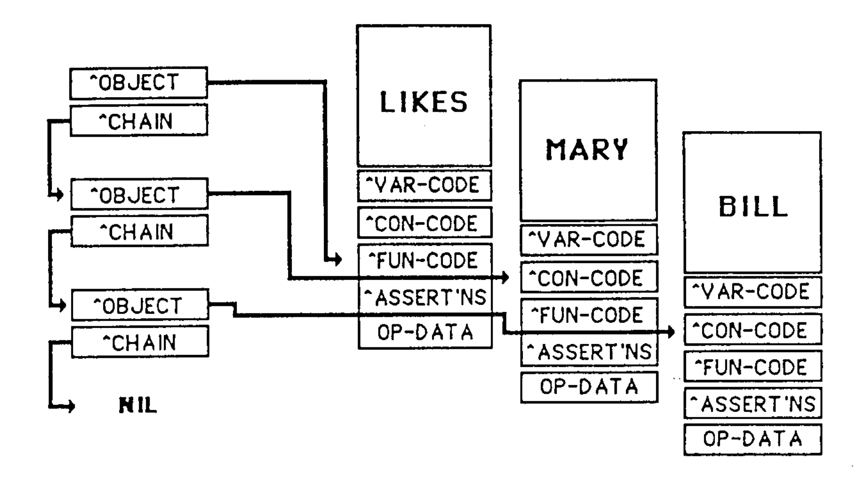
During unification, all substructures of the goal and all substructures of the clause head are associated with the indices of their parent structures. Saving a variable binding consists of saving both the pointers and indices of the variable and the object it is bound to. The binding stack is where these variable bindings are stored, and each element of the stack is thus an 8-byte data structure (**^var var.index ^obj obj.index**).

Variables are de-referenced by the Forth word `@BINDING`, which just searches the binding stack for a given variable name–index pair and recurses if the variable is bound to yet another variable.

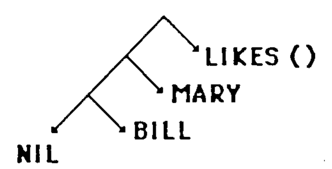
Control

A Prolog computation needs to keep track of both the clause it is executing and any alternative clauses (in case backtracking is necessary), of which subgoal to execute next on a successful exit from the current subgoal, and of all variable bindings in effect when the clause was called. This information constitutes the state of the computation. We represent the state of a computation by a

More general complex terms such as predicates (logical functions) are represented as chains in which the first object pointer points to the function-code field of a named object. Thus, the predicate **likes(mary,bill)** is represented internally as:



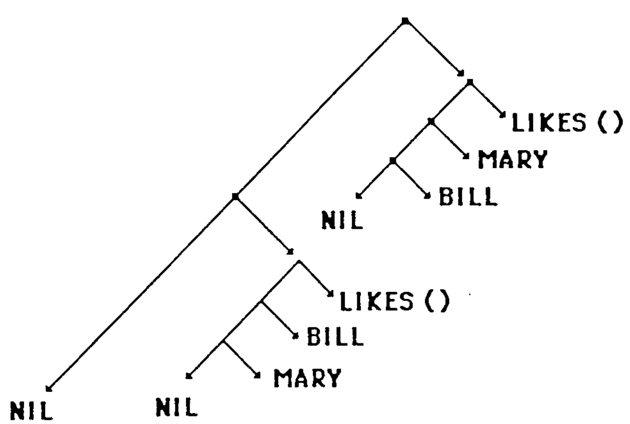
The tree representation of the predicate **likes(mary,bill)** is:



where the pair of parentheses denotes that the object pointer points to the function-code field of the named object.

Clauses

First some terminology: for the clause **A :- B,C,D.**, **A** is the head of the clause, **:-** is the neck, and **B,C,D** is the body of the clause. Clauses are represented internally as lists of terms, either simple or complex, where the first term in the list denotes the head of the clause. The remainder of the list is the body of the clause. Thus, the clause **likes(mary,bill) :- likes(bill,mary)**. may be represented by the tree:



The main functor of a clause is the functor (function name) of the head of the clause. A list of all clauses with the same main functor is an assertions list (e.g., a list of all the clauses whose functor is **likes**). The assertions field of the main functor points to this list. Thus (using a mixed representation), the list of clauses

6-word structure (called a goal frame). Each such state corresponds to a goal in the computation. The fields of the goal frame are:

Name	Contents
INDEX.N	next index
<ENV>	environment pointer
ASSERTIONS	assertions list pointer
GOALS	goal list pointer
INDEX.C	current index
REST.GOALS	indirect goal list pointer

The indices INDEX.N and INDEX.C (next index and current index respectively) are used in associating variables with the frame they were bound in, as described in the previous section on variable bindings. The environment pointer <ENV> points into the binding stack and indicates the binding environment on entry into the frame so the environment can be restored on backtracking. The GOALS pointer points to the goals list, a subset of the subgoals of the parent goal frame of the current goal frame. The GOALS list is used to determine the next goal if the current one is successful. The REST.GOALS pointer points into the goal frame of the parent; specifically, it points at the GOALS field of the parent frame. REST.GOALS is used to determine the next goal on exhausting the GOALS list in the current frame. The ASSERTIONS pointer points to the assertions list at the clause that was most recently resolved successfully with the first goal in the goals list. This is used to keep track of the remaining alternatives.

Given a goal frame (call it the parent frame) whose GOALS pointer points to some list of goals, the goals in the list are executed as follows. The first goal of the parent frame's goal list is made the current goal, and a pointer to it is stored in the variable GOAL (GOAL is a global variable, not part of the goal frame). The assertions list for the current goal is obtained from the assertions field of the main functor of the goal, and an attempt is made to resolve the current goal with the heads of the clauses in the assertions list. If this attempt is successful for some assertion in the list, then a goal frame is created (allocated on the backtrack stack), saving a pointer to the current binding environment in <ENV> and a pointer to the GOALS field of the parent frame in the REST.GOALS field. A pointer into the assertions list at the successfully resolved clause is saved in the ASSERTIONS field. If the computation ever has to backtrack to this point, it can then pick up the process of resolving the current goal with the assertions list at the point where it left off.

Assuming the resolution step is successful, the GOALS field of the new goal frame is filled with a pointer to the list of terms comprising the body of the clause just resolved (i.e., the body of the clause becomes the current goals list). If the body of the clause is empty (there are no subgoals), then the GOALS field of the current goal frame is filled with a pointer to the rest of the goals in the goals list of the parent frame. In other words, if there are no subgoals to consider, then the current goal (the first goal in the goal list of the parent frame) has been satisfied so the next step is to try and satisfy the remaining goals in the goal list of the parent frame.

If the attempt to resolve the current goal with the head of a clause in the assertions list fails for all clauses in the list, then the computation backtracks to the most recent goal frame that has remaining alternative ways to satisfy its goal. The computation proceeds until all the goals in the initial list have been satisfied (success) or until all the ways to satisfy the first goal in the initial list have been exhausted (failure).

The resolution and backtracking steps described are incorporated in the Forth words RESOLVE.FUN and RESOLVE.SINGLE. These words are the Prolog interpreter. RESOLVE.SINGLE is a special version of RESOLVE.FUN used to handle goals that have no arguments. The code addresses of these words are stored in the function-code and constant-code fields respectively of each named object (constants or variables). A Prolog procedure is called by placing a pointer to the appropriate field of a Prolog constant on the return stack and then exiting. The Forth inner interpreter then takes this return address as a pointer into the parameter field of some Forth word and so proceeds to execute either RESOLVE.FUN or RESOLVE.SINGLE, with the return stack containing a pointer that can be used to retrieve the assertions list.

Forth Interface

Built-in predicates are implemented in Forth, and use the predicate **builtin** as the interface between Prolog and Forth. **builtin** takes the name of a Forth word as its single argument and is distinguished from other predicates in that the function-code field of **builtin** does not contain the cfa of the word RESOLVE.FUN. Instead, control transfers directly to the Forth word that is the argument of the **builtin** call. This predicate is available for the user to access any of the underlying Forth system.

The only discipline required for user-defined built-ins is that the Forth word called drops the top of the return stack on entry and then exits by calling either $TRUE or $FALSE, which indicate, respectively, success or failure of the Prolog goal. Several Forth words are provided for parameter passing between Forth and Prolog. These words either retrieve the bindings of variables in the clause head or unify terms with variables in the head.

Invoking Prolog from Forth

Starting Up

Prolog is invoked by the Forth word PROLOG. The backtrack and binding stacks are initialized, and the Prolog interpreter is given the goal **prolog**, which implements a top-level read-execute-print loop:

```
prolog :- 'repeat ?-',read(X),execute(X).

    execute(X) :-
        (call(X),'print answer',tab(1),get0(Y),Y\=59,
        nl,display(yes) ;
        nl,display(no) ),
        !,fail.
```

The procedure **'repeat ?-'** is just a Prolog repeat which prints the **?-** prompt. The procedure **execute** takes the input goal and calls it. If the goal fails, **no** is output, **execute** fails and the prompt is repeated. If the goal succeeds, **'print answer'** prints the variable bindings and then waits for a single character input. If the character is ; , we backtrack to **call** and try to satisfy the goal in another way. Otherwise **yes** is output, **execute** fails and the prompt is repeated.

A read-assert loop is entered by calling **user**:

```
user :- repeat,read(X),(X=stop;assertz(X),fail).
```

For example, the input text block that follows adds three clauses to the data base between **user** and **stop**. Following **stop** the user is back in the read-execute-print loop.

```
?- user.
likes(bob,mary).
likes(mary,X) :- is_funny(X).
is_funny(bob).
stop.
?-
```

In contrast to most Prolog implementations, variables retain their names in the internal representation of clauses. The negative aspect of this feature is that variable names take up space in the dictionary. One interesting experiment would be to implement predicates that create and switch Forth vocabularies. This would permit different Prolog data bases to exist in different vocabularies. Because of Forth we can get "hierarchical multiple worlds" for free.

Size, Speed and All That

The entire interpreter, including the parser but exclusive of the stacks, takes up on the order of 10K bytes. Running entirely in Forth (MicroMotion MasterForth on the Apple Macintosh), the interpreter will do about 22 LIPS (Logical Inferences Per Second, which effectively measure the procedure call rate) as timed with naive reverse. By recoding just the word @BINDING in assembler, there is a threefold increase in speed, and more efficiency can be gained by recoding other words in the inner loop of the Prolog interpreter. There is, thus, every indication that, with tuning, this implementation could be made competitive in speed with other Macintosh implementations of Prolog (see [PIE87] for a comparison of four Macintosh Prolog products). The version used in our application [PAL87] is written in polyFORTH running on a PC. No performance measurements have been made on the PC version.

One of the inefficiencies in the space usage of this implementation comes from not reclaiming goal frames. This means that the backtrack stack contains a goal frame for every inference that has been made, thereby saving more than is absolutely necessary for backtracking. A goal frame could be reclaimed on return (i.e., when GOALS list has only one element) if its ASSERTIONS list has only one element (no more alternatives).

Other Implementations

Several other implementations of Prolog in Forth have been mentioned in the literature. Harris has described a Prolog interpreter that has been used in space-related work [HAR86]; however, few details of this implementation have been published. The most extensive treatment is by Townsend and Feucht [TOW86] who present a very good discussion of the Prolog interpreter mechanism.

Their implementation of binding environments is similar to that described here, and both are derived from the simple schemes used in the primitive LISP implementations of Prolog [NIL84].

It should be noted that the Prolog syntax used in [TOW86] is nonstandard, and no mention is made of the implementation of the standard Prolog built-in predicates; however, note that writing the Prolog parser is a major effort, as is writing a complete set of built-ins. In our implementation there are an equal number of screens (approximately 30) devoted to the interpreter, the parser and the built-in predicates.

Summary of Evaluable Predicates

The following is a list of the built-in predicates provided in this implementation, each with a brief description of its semantics. The majority of the Clocksin and Mellish predicates are included. Greater detail can be found in [CLO81].

arg(N,T,A)	The Nth argument of term **T** is **A**.
asserta(C)	Assert **C** as first clause.
assertz(C)	Assert **C** as last clause.
atom(T)	Term **T** is an atom.
atomic(T)	Term **T** is an atom on an integer.
call(P)	Execute the Prolog procedure call **P**.
builtin(W)	Execute the Forth word W.
clause(P,Q)	There is a clause with head **P** and body **Q**.
consult(N0,N1)	Extend program with clauses from screen N0 thru N1.
display(T)	Display term **T** on terminal.
fail	Backtrack immediately.
functor(T,F,N)	The top functor of term **T** has name **F**, arity **N**.
get(C)	The next non-blank character input is **C**.
get0(C)	The next character input is **C**.
halt	Halt Prolog, exit to Forth.
integer(T)	Term **T** is an integer.
Y is X	**Y** is the value of the arithmetic expression **X**.
listing(P)	List the procedure(s) **P**.
name(A,L)	The name of the atom **A** is string L.
nl	Output a new line.
nonvar(T)	The term **T** is a non-variable.
not(P)	Goal **P** is not provable.
op(P,T,A)	Make atom **A** an operator of type T, precedence **P**.
put(C)	The next character output is **C**.
read(T)	Read term T.
repeat	Succeed repeatedly.
retract(C)	Erase the first clause of form **C**.
skip(C)	Skip input characters until after character **C**.
tab(N)	Output **N** spaces.
trace	Start tracing.
true	Succeed.
untrace	End tracing.
var(T)	Term **T** is a variable.
write(T)	Write the term **T**.
!	Cut any choices taken in the current procedure.
X < Y	As numbers, **X** is less than **Y**.
X =< Y	As numbers, **X** is less than or equal to **Y**.
X > Y	As numbers, **X** is greater than **Y**.
X >= Y	As numbers, **X** is greater than or equal to **Y**.
X = Y	Terms **X** and **Y** are unified.
X \= Y	Terms **X** and **Y** are not unified.
T =.. L	The functor and args of term **T** comprise the list L.
X == Y	The terms **X** and **Y** are strictly identical.
X \== Y	The terms **X** and **Y** are not strictly identical.

The following is a list of the Clocksin and Mellish predicates that have not been implemented, each with a brief description of its semantics.

debugging	List all current spy points.
nodebug	Remove all current spy points.
nospy P	Remove spy point from predicate **P**.
see(X)	Open file **X** for input.
seeing(X)	File **X** is open for input.
seen	Close file **X** for input.
spy P	Set a spy point on predicate **P**.
tell(X)	Open file **X** for output.
telling(X)	File **X** is open for output.
told	Close file **X** for output.

References

[CLO81] Clocksin, W. F., and Mellish, C. S. 1981. *Programming in Prolog*. New York: Springer-Verlag.

[HAR86] Harris, H. M. 1986. Development of an expert system for command and control of an orbiting spacecraft. *J. Forth Appl. and Res.* 4(2):305.

[NIL84] Nilsson, M. 1984. The world's shortest prolog interpreter? In *Implementations of Prolog*, ed. J. A. Campbell. Sussex, England: Ellis Horwood.

[PAL87] Paloski, W. H., Odette, L. L., Krever, A. J., and West, A. K. 1986. Use of a Forth-based Prolog for real-time expert systems. I. Spacelab life sciences experiment application. *J. Forth Appl. and Res.*, this issue.

[PIE87] Pierson, D. L. 1987. AI: four Prologs for the Macintosh. *Dr. Dobb's Journal of Software Tools* 12(4):30-41.

[TOW86] Townsend, C. and Feucht, D. 1986. *Designing and programming personal expert systems*. Blue Ridge Summit, PA: TAB Books.

■

(suite de la page 8)

DANS LA CREATION DE MANUEL F83. JE N'IGNORE PAS QU'IL DOIT ETRE ECRIT AVEC WP. AYANT LA CHANCE DE POSSEDER UNE IMPRIMANTE POSTCRIPT DANS MA BOITE. JE PEUT ME PERMETTRE DE VOUS METTRE EN PAGE L'EPREUVE DE VOTRE MANUEL. JE CONNAIS TRES BIEN WP 4.2 ET 5.0. UN FORTHIEN QUI VOUS VEUT DU BIEN...
BIEN RECU VOS SUGG SUR MON IDEE DE DEMO JE VOUS TIENDRAIS AU COURANT DANS QUELQUES JOURS SUR MES IDEES DANS LE DE-TAIL. ATCHAO .....

F32 13.10.88 11h40
POUR CET ESSAI DE TF NOUS EMULIONS LE 86 AVEC UN 68020 A 25 MHZ, UN 68881 POUR FONCTIONS MATH, EN INTERPRETANT LE CODE INTEL. CA TOURNE A PEU PRES AUSSI VITE QUE LE PC DE BASE

EUREKA 13.10.88 21h33
REPONSE A F32: IL EXISTE A MA CONNAISSANCE 3 FORTH POUR ATARI ST:
- MT/FORTH DE DATA BECKER & ABACUS SOFT
- UNIX FORTH EN DOMAINE PUBLIQUE
- HOLMES & DUCKWORTH FORTH (H&D FORTH)
MALHEUREUSEMENT ILS NE SONT DISPONIBLES QU'EN GRANDE BRETAGNE! SOLUTION: ALLER ACHETER UNE REVUE ST ANGLAISE (WH SMITH, RUE DE RIVOLI PARIS PAR EX) PRENDRE LES REFERENCES DES SOCIETES QUI LES VENDENT ET COMMANDER. AUTRE SOLUTION: DEMANDER UNE COPIE DES SOFT EN DOM PUB AUPRES DES ATARISTES

SECRETAIRE 14.10.88 12h21
FORTH ATARI: ATTENTION, N'OUBLIEZ PAS QUE NOUS DIFFUSONS VOLKS'FORTH POUR ATARI.

SECRETAIRE 17.10.88 16h24
L'ADHERENT LAURENT MANGANE RECHERCHE UN SYSTEME FUTURSYS (MEME D'OCCASION) ET SURTOUT D'OCCASION VU QUE NEUF, BEN CE SERAIT UN PEU DUR... CONTACT:
LAURENT MANGANE 39, RUE HENRI GOURMELIN
91200 ATHIS MONS TEL DOM:69.38.64.01

JMCFORTH 18.10.88 10h24
QUELQU'UN A-T-IL DEVELOPPE UN PROGRAMME DE CALCUL EN BCD SUR PLUS DE 15 CHIFFRES SIGNIFICATIFS AVEC CONVERSION DE CHAINE ALPHANUM. EN NOMBRES BCD OU ALORS QUELQU'UN CONNAIT-IL UN LIVRE QUI EN PARLE?

SECRETAIRE 18.10.88 11h51
A l'attention des TELECHARGEURS forcenes. On vient de monter les nouveaux softs sur l'option 7:
- dBASE: MENU DEROULANT
- TFORTH: RECORD.FTH IMPRIM.FTH FWIND.FTH FWMENU.FTH FWMENU.OVL

SECRETAIRE 20.10.88 11h14
COMMENT CONCATENER UN CODE DE CONTROLE: dans une variable chaine alphanumerique sous TURBO-Forth? Definir d'abord:

```
  : CODE$ CREATE C, DOES> 1 ;
Puis compiler un code, exemple:
  7 CODE$ PING
Et maintenant:
  80 STRING A$
  " CECI EST UN BIP:" A$ $!
  PING A$ APPEND$
  A$ TYPE     affiche A$ et emet un BIP sonore!
```

CYRILLE 21.10.88 17h06
JE CHERCHE TOUT SORTE DE LOGICIEL SOUS PC. LAISSE MOI UN MESSAGE DANS MA BAL CYRILLE MERCI D'AVANCE

GUILLAUMAUD 26.10.88 08h08
BONJOUR, JE TROUVE QUE LE SERVEUR DE JEDI EST BIEN DESERTIQUE! QUE CE PASSE-T-IL? SERAI-CE LA PERIODE D'HIBERNATION DES FORTHIENS? OHE DU SERVEUR, IL Y A QUELQU'UN? SACHEZ QUE LE-FORTH V.2.00 SERA DISPONIBLE DEBUT 89. SALUTATIONS

SECRETAIRE 27.10.88 09h29
NOUVEAUX OPERATEURS DE PILE: Vous connaissez tous les DUP, DROP, SWAP, OVER, ROT, PICK, ROLL etc... Dans le dernier

numero de Vierte Dimension, Rolf Kretzschmar propose une normalisation des denominations pour substituer a ces operateurs un systeme excluant toute ambiguite:

```
DROP  devient  1#X
DUP   devient  1#K
SWAP  devient  2#P
2DROP devient  2#2X
```

Ca a l'air de compliquer les choses me direz vous. Pas tellement en regard de la logique apportee au traitement. Prenons le premier caractere de ces nouveaux operateurs 1#X, soit 1: le 1 signifie qu'il faut pointer sur le premier element a partir du sommet de la pile de donnees. Ensuite, le signe # est un separateur. Enfin, la lettre X indique qu'il faut supprimer la donnee. Trois lettres sont reservees:

```
X pour suppression (ex: DROP)
P pour deplacer    (ex: SWAP ROT)
K pour copier      (ex: OVER PICK)
```

Cette lettre peut etre precedee d'un parametre indiquant le nombre d'elements a traiter. Exemple:

5#3K

Signifiera qu'il faudra copier 3 elements a partir du 5eme nombre depose sur la pile de donnees:

```
a b c d e 5#3K
!5eme element
===== 3elements
```

donne:

```
a b c d e a b c
```

Pour les principaux operateurs de pile, voici les equivalents en notation N#X N#K et N#P:

```
DROP  alias 1#X  ! DUP   alias 1#K
NIP   alias 2#X  ! SWAP  alias 2#P
OVER  alias 2#K  ! 2DROP alias 2#2X
2DUP  alias 2#2K ! ROT   alias 3#P
-ROT  alias 3#2P ! 2SWAP alias 4#2P
2OVER alias 4#2K ! PICK  alias n#K
```

Le dernier cas, PICK prend un element sur la pile pour effectuer la manoeuvre. Dans l'etat actuel de l'interpreteur FORTH, si l'on souhaite appliquer cette technique, consiste a redefinir les operateurs comme suit:

```
CODE 1#K ( n1 --- n1 n1 )
AX POP   AX PUSH   1PUSH   END-CODE
```

et plus loin:

```
: DUP STATE @
IF [COMPILE] 1#K
ELSE 1#K THEN ; IMMEDIATE
```

Sinon, la modification de l'interpreteur interne nous contraint a analyser les mots en cas d'echec de recherche dans le dictionnaire. Exemple: la recherche de 6#3P dans le dictionnaire echoue; passage par une routine d'analyse syntaxique de 6#3P similaire a NUMBER mais appliquee aux manipulateurs de pile nouvelle formule.

Cette solution est passablement complexe, car il faut l'optimiser tant en interpretation qu'en compilation.

SECRETAIRE 28.10.88 09h30

TEST PRESENCE IMPRIMANTE 0: Avec un BIOS compatible IBM et une imprimante parallele, on peut tester l'etat de l'imprimante n0 avec le mot suivant:

```
DECIMAL  CODE (PR-STAT)  2 # AH MOV   0 # DX MOV
   23 INT   144 # AH XOR
   0= IF  -1 # AX MOV   ELSE   0 # AX MOV
   THEN   1PUSH   END-CODE
' (PR-STAT) IS PR-STAT
```

Dans ce cas PR-STAT ne delivre un flag vrai que si l'imprimante est connectee et prete a imprimer.

SECRETAIRE 28.10.88 09h33

LECTURE DE n CARACTERES D'UN FICHIER: la lecture s'effectue a la position courante du pointeur de fichier; celui-ci est deplace en avant de len' octets pour poursuivre une lecture sequentielle.

```
: LIT-DEBUT  ( <fichier> -- )
   FILENAME 0 (OPEN) ?DOS-ERR >R
   DSEGMENT PAD 10 R@ (GET) ?DOS-ERR
   PAD SWAP TYPE R> (CLOSE) ;
```

LIT-DEBUT ESSAI affiche les 10 premiers caracteres de ESSAI.FTH

SECRETAIRE 28.10.88 09h35

COMPTER LES FICHIERS FORTH DANS LE repertoire courant:

```
: COMBIEN-DE-FICHIERS ( <masque> -- )
  0   PATHWAY 0 (SEARCH0)
  IF BEGIN 1+ (SEARCH)  NOT  UNTIL
  THEN . ;
```

COMBIEN-DE-FICHIERS B:\FORTH*.FTH affiche le nombre de fichiers repondant au masque '*.FTH' dans le repertoire FORTH de B:

SECRETAIRE 28.10.88 09h37

EFFACEMENT SELECTIFS DE FICHIERS A PARtir de TURBO-Forth: voici un mot permettant d'effacer dans le repertoire courant un groupe de fichiers (ce que n'autorise pas DEL) avec selection au clavier des fichiers a effacer:

```
: ERA    ( <masque> -- )
  0   PATHWAY 0 (SEARCH0)
  IF  BEGIN  CR .NAME ." Effacer ? O/N
        KEY UPC DUP EMIT ASCII O =
        IF  DMA 30 + 65  BDOS ?DOS-ERR 1+
        THEN  (SEARCH) NOT  UNTIL
  THEN CR . ." fichiers effaces" ;
```

ERA *.* affiche successivement les fichiers qu'il propose d'effacer puis affiche le nombre total de fichier effaces.

SECRETAIRE 28.10.88 09h48

FONCTION VIEW EN TURBO-FORTH. Voici un exemple d'utilisation conjointe des mots OPEN HANDLE GETLINE .LINE et CLOSE pour definir un mot VIEW listant un fichier a partir de la rencontre d'un mot quelconque. La syntaxe en est:

VIEW <fichier> <mot>

et comme la premiere occurence d'un mot Forth dans un fichier est generalement sa definition (hors commentaires), on utilisera VIEW pour retrouver le source d'un mot puis voir son utilisation dans le reste du fichier. La commande VIEW <fichier> seule est equivalente a LIST <fichier>:

```
: VIEW   ( <fichier[.ext]> <mot> -- )
 ECHO   @  \ conserve etat de ECHO
 HANDLE @  \ conserve HANDLE courant
 ECHO OFF  \ pas d' ECHO pour commencer
 OPEN      \ ouvre le fichier
 BL WORD DROP  \ isole mot a chercher du flot actuel
 HANDLE !  \ attribue ticket couran au fichier
 BEGIN     \ commence a lire fichier:
   GETLINE  \ prend une ligne
   ECHO @ 0=  \ ECHO encore OFF ?
   IF HERE COUNT  \ si oui: chaine du mot a chercher
      BUFFER COUNT \ chaine chargee dans le tampon
      SEARCH  \ rech. mot dans tampon
      NIP     \ position mot inutile
      IF .LINE  \ si trouve: affiche la ligne
         ECHO ON  \  et on continue en ECHO ON
      THEN  \ si pas trouve: on continue comme ca
   THEN \ si non: continue sans chercher le mot
   EOF? @ STOP? OR \ jusqu'a fin
        \ fichier ou stop au clavier
 UNTIL   \ c'est fini:
 CLOSE   \ on ferme le fichier
 HANDLE !  \ on retablit le HANDLE
 ECHO   !  ; \ on retablit ECHO
```

SECRETAIRE 28.10.88 09h56

CHANGER L'EDITEUR DE TURBO-Forth: TURBO-Forth permet d'utiliser votre editeur habituel en remplacement de celui fourni sur la disquette contenant le module 1. Si votre editeur n'accepte pas plus d'un parametre (celui du fichier a editer), il faudra retrouver l'erreur en vous aidant du message de localisation delivre par l'interpreteur. Si votre editeur ac cepte de se positionner au lancement sur une ligne, ou mieux sur une ligne et une colonne, selon une syntaxe differente, il est preferable de redefinir le mot EDIT de facon appropriee en construisant dans la variable chaine COMMAND$ la chaine de parametres pour votre editeur (apres une erreur le mot (WHERE) charge dans COMMAND$ le nom du fichier incrimine):

```
: EDIT  ( -- )
  DECIMAL COMMAND$ NIP
  IF   " /L="       COMMAND$ APPEND$
   IN-LINE @ (.) COMMAND$ APPEND$
   " /C="       COMMAND$ APPEND$
   IN-CHAR @ (.) COMMAND$ APPEND$
  ELSE CR ." Fichier a editer:"
COMMAND$ INPUT$
  THEN " PROGRAM EDT.EXE" $EXECUTE ;
```

EDIT est ici redefini pour un editeur .externe EDT.EXE acceptant une syntaxe de lancement:

EDT [fichier][/L=xxx][/C=yyy avec xxx et yyy pour la ligne et la colonne du curseur ou debuter l'edition

---

LORD SAM 28.10.88 12h17
Bonne nouvelle pour les Forthiens atarimaniaques: le telechargement se met en place pour le ST. Allez donc faire un tour sur SAM*ATA et recopiez le listing de la partie reception... Attention ce n'est pas en Forth mais en GFA (he oui).... Quelques fichiers sont deja disponibles mais le gros de la troupe attend encore quelques jours pour montrer le bout de son nez. Les fichiers JEDI seront bien sur de la partie, et vous pourrez telecharger indifferemment du source ou de l'executable avec un debit de 3Ko/mn.

Pour repondre a TICK , il est en effet possible de commander du soft aux States en indiquant son numero de VISA card et l'expiration date... Attention, aux USA on ne plaisante pas avec les escroqueries a la carte bleue et le mode de paiement y est donc tres repandu. J'ai personnellement deja commande quelques softs directement aux States par ce moyen: un coup de fil le mardi soir, le paquet etait dans ma boite le vendredi matin. Et pas de mauvaise surprise, le double du bordereau VISA accompagne le paquet, avec "TOLL ORDER" en guise de signature (commande telephonique)...

SECRETAIRE 03.11.88 10h10
FICHIER BATCH: SAUTER 1 LIGNE pour pouvoir faire afficher une ligne blanche par un programme batch, il suffit d'utiliser la commande echo immediatement suivie d'un point:

```
    ECHO OFF
    ECHO.
    ECHO TITRE DU PROGRAMME
    ECHO.
    ECHO.
    ECHO SUITE DU PROGRAMME
```

ATTENTION: ne fonctionne que sous ms-dos version 3. et suivantes. Mais il y a mieux que 'ECHO.'. Il suffit de taper ECHO <Alt 255>.

SECRETAIRE 03.11.88 10h13
SOUS PROGRAMME BATCH. si dans un programme batch vous faites appel a un autre programme batch, a la fin de l'execution du programme appele la main est rendu a ms-dos et non au programme appelant. Pour remedier a ce probleme, faire preceder l'appel du programme par la commande: command /c. exemple: DEMO.BAT

```
  ECHO OFF
  COMMAND /C AFFICHE (S/PROG)
  REM SUITE DU PROGRAMME DEMO
    ...
```

SECRETAIRE 03.11.88 10h15
CONTROLE EXTERNE DU PC: pour passer les commandes a un utilisateur minitel, il suffit de posseder un minitel 1b, de savoir le configurer ou de posseder une carte modem standart v23 (ou un minitel retournable) et de taper sous dos:

```
  MODE COM2:1200,E,7,1
  CTTY COM2
```

Pour redonner le contole du pc a l'utilisateur, il faut que le minitelliste tape CTTY et <CR> ou touche SHIFT-ENVOI.
Ndlr: ce truc a été trouvé sur 3614 MICROB mais apparement ne fonctionne pas!!!

SECRETAIRE 03.11.88 10h41
LA GERANCE DE SAM VIENT DE RAJOUTER 21 PROGRAMMES EN PASCAL DANS LE TELECHARGEMENT. ATTENTION, LORS DE LA SELECTION D'UN PROGRAMME A TELECHARGER, UN BUG OBLIGE UNE DOUBLE SAISIE. EX:
 CHOIX FORTH,
 CHOIX RSINIT (PROG No7), TAPER:
  7 ENVOI
  7 ENVOI a nouveau
CECI SEULEMENT POUR LES PROGS EN PREMIERE PAGE. TOUT EST NORMAL POUR LES PROGS EN SECONDE PAGE. CE BUG N'EST PAS DE NOTRE FAIT.

JMCFORTH 03.11.88 14h58
CHERCHE ROUTINES EN FORTH 83 DE MATH EN BCD AVEC AU MOINS 15 CHIFFRES SIGNIFICATIFS AINSI QUE LA CONVERSION D'UNE CHAINE ALPHANUMERIQUE EN NOMBRE BCD ET INVERSEMENT. A L'AT-TENTION DU SECRETAIRE: COMMENT MANIPULER PLUS DE DEUX FI-CHIERS DE DONNEES EN TURBO FORTH NOTAMMENT DES FICHIERS DIRECTS

FORTH7 06.11.88 15h48
Reponse a JMCFORTH: TF83 admet l'ouverture simultanee de nombreux fichiers: pour cela il faut
1) placer dans CONFIG.SYS une ligne FILES=14
2) redimensionner la constante FILES 14 is FILES
14 permet ici d'ouvrir 10 fichiers car les tickets 0 * 4 sont reserves aux peripheriques standards. 14 tampons sont alors accessibles pour MS-DOS et pour FORTH (adresse donnee par BUFFER). Les tampons Forth sont de 256 cars. Pour le travail en acces direct, il peut etre necessaire d'en definir de plus grands en intra ou extra segment. (Precisez votre probleme)

FORTH7 06.11.88 16h29

```
\ Copies de fichiers avec SHELL
30 STRING COPYING
: COPY ( <arg1> <arg2> -- )
       " COPY " COPYING $!
   BL WORD COUNT COPYING APPEND$
          " " COPYING APPEND$
   BL WORD COUNT COPYING APPEND$
        " > NUL" COPYING APPEND$
               COPYING SHELL ;
```

COPY s'utilise exactement comme sous MS-DOS:
  COPY A:*.FTH C:\FORTH
L'ajout '> NUL' est une astuce pour supprimer l'affichage des fichiers par COPY (reutilisable dans vos .BAT)

GUILLAUMAUD PH. 07.11.88 08h07
BONJOUR, MODIFICATION DE L'ENVIRONNEMENT DU DOS DANS UN FICHIER BATCH. EX.:

```
SET OLDPATH=%PATH%
SET PATH=C:\TURBO\FORTH;
REM ... suite du programme
SET PATH=%OLDPATH%
SET OLDPATH=
```

LA SYNTAXE "SET Var.=%VarE%" NE MARCHE QUE DANS UN BATCH. CECI PERMET D'AFFECTER A Var LE CONTENU DE VarE.

GUILLAUMAUD PH. 10.11.88 08h14
Bonjour, Dans la serie des TRUCS du DOS: recherche d'un fichier quelque soit sa position sur le disque; voici le Programme TROUVE.BAT:

CHKDSK /V ¦ FIND "%1"
Le symbole "¦" s'obtient par appui sur ALT+124. Ceci a l'avantage de ne pas necessiter d'utilitaire ...

FORTH7 11.11.88 14h42
*** TELECHARGEMENT ASCII SAM*JEDI ***
Vous etes tellement fauche que vous n'avez pas commande LCECOM a JEDI. Pour telecharger nos fichiers ASCII, y'a un moyen bestialement simple sous ms-dos: branchez votre cable RS232-MTL recupere d'un soft quelconque de telechargement puis initialisez en mode minitel:
     MODE COM1: 1200,E,7,1
(MODE.EXE est sur votre disk system) procedez a la demande de telechargement jusqu'au message 'tapez ENVOI' puis avant de faire cet ENVOI sur le MNTL faites
     COPY COM1: <fichier>
Reprenez un peu <fichier> a l'editeur c'est pas plus complique que ca!

FORTH7 23.11.88 19h03
Eh! C'est mon dernier message qui vous la coupe? Faites un peu le BREAK: voici comment gerer le Break en Turbo-Forth:

```
\ Vectorisation de Ctrl-Break sur WARM
ONLY FORTH DEFINITIONS ALSO HEX
CODE  BREAK ( -- ) \ point d'arret
      STI   20 # AL MOV   20 # AL OUT
      0103 #) JMP   END-CODE
CODE  SETBREAK ( -- ) \  vectorise INTs
      ' BREAK >BODY # DX MOV
      2523 # AX MOV   21 INT    1 # DL MOV
      3301 # AX MOV   21 INT    NEXT END-CODE
SETBREAK FORGET SETBREAK DECIMAL
```

Ctrl-C et Break ne sortent plus de TF: tres utile pour stopper un INCLUDE qui n'en finit pas...

F32 28.11.88 10h08
L'AUTEUR ANONYME DE CE BEL ARTICLE SUR LA GESTION DE FENETRES PARU DANS LE DERNIER NUMERO DE JEDI EST INVITE A CONTACTER F32; SON INTERET POUR LES ARBRES ET GRAPHES EST DE BON AUGURE...

A PARAITRE PROCHAINEMENT DANS JEDI: STRUCTURE DU PROCESSEUR F32 ET MODE DE FABRICATION DU JEU D'INSTRUCTIONS. PORTEZ-VOUS BIEN !

SECRETAIRE 02.12.88 16h10
FINI LES GREVES DE LA POSTE...? SI VOUS DOUTEZ DE LA DILIGENCE DE VOTRE FACTEUR, OPTEZ POUR LA SOLUTION TELECOPIEUR. MAINTENANT, POUR 8500 FR HT AYEZ LE VOTRE. CONTACT:
STE SHINING 24 CITE TREVISE 75009 PARIS
TEL: 42.47.05.81 OU TELECOPIE: 42.47.16.89
A CE PRIX LA... ON SERAIT TENTE, NON?

SECRETAIRE 05.12.88 09h36
MESSAGES DANS LE FORUM: Je rappelle que le FORUM est a votre disposition pour poser toutes les questions concernant la programmation et l'utilisation du langage FORTH. Nous repondrons a vos questions dans les meilleurs delais.

D'autre part, si vous avez des petits trucs a communiquer, n'hesitez pas a en faire profiter autrui, car vous serez bien contents quand vous-meme profiterez des astuces d'autrui.

Enfin, si vous redigez un message a usage confidentiel, vous pouvez le transmettre a plusieurs correspondants: apres composition et validation, indiquez le numero de chaque correspondant+ENVOI.

PATRICKS 08.12.88 16h40
QUI PEUT ME DIRE COMMENT TELECHARGER AVEC LE LOGICIEL SPTEL EN UTILISANT KERMIT MERCI.

SECRETAIRE 08.12.88 21h39
REPONSE A PATRICKS: LE LOGICIEL SPTEL NOUS EST INCONNU. MAIS POUR TELECHARGER LES PROGS DE SAM*JEDI, IL SUFFIT DE DISPOSER D'UN PROGRAMME ASSURANT UNE 'CAPTURE' EN MODE ASCII, CECI POUR LES SOFTS FORTH ET PROLOG. POUR LES AUTRES, EQUIPEZ VOUS DE TELECHAR ET QUI EST PROPOSE EN DEBUT DE CONNEXION (COMMANDE KERMIT=KERMIT ET TELECHAR). A+

F32 11.12.88 00h07
-------------------- F32 --------------------
LA SPEC PRELIMINAIRE EST SOUS PRESSE; SORTIE DANS LE PROCHAIN JEDI. IL Y AURA DESCRIPTION DE LA STRUCTURE, UN (PETIT) NOYAU DU (GRRRAND) JEU D'INST.
AFFUTEZ VOS MENINGES POUR DONNER AU MODELE DE QUOI S'EXERCER. RESTRICTIONS CONNUES:
-64K ADRESSES SUR CHAQUE PILE

SECRETAIRE 12.12.88 09h20
COMMENT LANCER DEBUG.COM depuis TF83: TURBO-Forth dispose bien d'un decompilateur, mais pas d'un desassembleur. Mais avec ce petit truc, vous allez pouvoir lancer DEBUG.COM (fourni avec la disquette MSDOS de votre systeme) puis revenir sous TURBO-Forth:

```
80 STRING A$  80 STRING B$
" A$ PASS PROGRAM DEBUG.COM " B$ $!
: CODE-DEBUG B$ $EXECUTE ;
```

Maintenant, tapez CODE-DEBUG pour lancer le desassembleur.

SECRETAIRE 12.12.88 09h22
DECODAGE DES TOUCHES: pour retoucher le contenu de KEYBFR.COM, il faut connaitre non seulement le code ASCII d'une touche, mais aussi son SCAN-CODE. C'est ce que fait ce petit proogramme:

```
HEX    CODE SCAN CODE
  0 # AH MOV   16 INT   1PUSH END-CODE
DECIMAL
: DECODE ( ---)
BEGIN  CR ." APPUI SUR UNE TOUCHE:"
  SCAN CODE   256 /MOD
     CR ." SCAN-CODE: "
  DUP CR ."      CODE: " .  DUP  32 >
IF     CR ."     ASCII: " EMIT
  ELSE  DROP    THEN
  CR CR ." AUTRE TEST (O/N) ?" KEY
  UPC ASCII O <> UNTIL ;
```

SECRETAIRE 12.12.88 10h51
TURBO-FORTH EN ALLEMAND DISPONIBLE: GRACE A LA TENACITE ET LA PATIENCE DE NOTRE CORRESPONDANT MUNICHOIS FRED BEHRINGER, NOUS DIFFUSONS MAINTENANT LE MODULE 1 DE TURBO-FORTH DANS LA LANGUE DE GOETHE.
POUR RAPPEL, LE MODULE 1 EXISTE EN:
- FRANCAIS
- ANGLAIS
- ALLEMAND
MAIS NOUS SOMMES TOUJOURS A LA RECHERCHE D'UN TRADUCTEUR BENEVOLE POUR UNE VERSION ESPAGNOLE. OUI, CERTES, L'ANGLAIS EST PEUT ETRE UNIVERSEL, MAIS FAISONS COMME LES AMERICAINS CHEZ QUI ON PARLE PLUS SOUVENT EN ITALFRANPORTUGANOL QU'EN PATOIS SHEKSPIRIEN....

SECRETAIRE 13.12.88 09h19
CODE TOUCHE CLAVIER SUR PORT A DU 8255: Pour rendre independant un 'KEY' de l'appui sur SHIFT, CTRL ou ALT, voici un KEY modifie qui va delivrer le code physique d'une touche clavier:

```
hex
: key-port ( --- c)
key drop 060 pc@ ;  decimal
```

L'execution de KEY-PORT depose 1 sur la pile quand on appuie sur ESC, 2 quand on appuie sur 1 ou &, et ceci que vous soyez en QWERTY ou AZERTY ou votre clavier reaffecte n'importe comment.

F32 13.12.88 13h05
JE FAIS DES RECHERCHES SUR LA STRUCTURE DU DICTIONNAIRE, ET IL ME FAUDRAIT CONNAITRE EN DETAIL LE FONCTIONNEMENT DU DECOMPILATEUR POUR EN BIDOUILLER UN SPECIAL. J'AI FAIT SEE SEE ET SEE (SEE) MAIS CA ME LAISSE UN PEU SEC. POUVEZ- VOUS M'ECLAIRER? JE SUIS SUR F83 CPC (ET OUI...) MAIS JE NE PENSE PAS QUE CA GENE. MERCI
REMARQUEZ QUE LE DICTIONNAIRE EST FAIT COMME UN ARBRE, JE VOUDRAIS FAIRE APPARAITRE LA STRUCTURE DE FACON SIMPLE

BYG INFO 14.12.88 11h32
BYG INFORMATIQUE TEL : 61 26 44 36
NOUS VOUDRIONS NOUS PROCURER LES SOURCES POURRIEZ VOUS NOUS INDIQUER LA MARCHE A SUIVRE SVP.

SECRETAIRE 14.12.88 14h00
REPONSE A BYG INFO: APRES PRECISIONS DEMANDEES PAR TEL, JE MET MA REPONSE SUR SAM*JEDI, A SAVOIR:
- LE SOURCE ASCII ET LA DOC KERMIT SERONT DIFFUSES SUR TELECHARGEMENT DANS LA NOUVELLE RUBRIQUE 'C'. EGALEMENT DES NOUVEAUTES EN TF83, dBASE ET C.